Coordinated Constraint Relaxation Using a Distributed Agent Protocol

Mohd Fadzil Hassan

Doctor of Philosophy

Centre for Intelligent Systems and their Applications (CISA)

School of Informatics,

The University of Edinburgh.

2007

Abstract

The interactions among agents in a multi-agent system for coordinating a distributed, problem solving task can be complex, as the distinct sub-problems of the individual agents are interdependent. A distributed protocol provides the necessary framework for specifying these interactions. In a model of interactions where the agents' social norms are expressed as the message passing behaviours associated with roles, the dependencies among agents can be specified as constraints. The constraints are associated with roles to be adopted by agents as dictated by the protocol. These constraints are commonly handled using a conventional constraint solving system that only allows two satisfactory states to be achieved – completely satisfied or failed. Agent interactions then become brittle as the occurrence of an over-constrained state can cause the interaction between agents to break prematurely, even though the interacting agents could, in principle, reach an agreement. Assuming that the agents are capable of relaxing their individual constraints to reach a common goal, the main issue addressed by this thesis is how the agents could communicate and coordinate the constraint relaxation process. The interaction mechanism for this is obtained by reinterpreting a technique borrowed from the constraint satisfaction field, deployed and computed at the protocol level.

The foundations of this work are the Lightweight Coordination Calculus (LCC) and the distributed partial Constraint Satisfaction Problem (CSP). LCC is a distributed interaction protocol language, based on process calculus, for specifying and executing agents' social norms in a multi-agent system. Distributed partial CSP is an extension of partial CSP, a means for managing the relaxation of distributed, over-constrained, CSPs. The research presented in this thesis concerns how distributed partial CSP technique, used to address over-constrained problems in the constraint satisfaction field, could be adopted and integrated within the LCC to obtain a more flexible means for constraint handling during agent interactions. The approach is evaluated against a set of over-constrained Multi-agent Agreement Problems (MAPs) with different levels of hardness. Not only does this thesis explore a flexible and novel approach for handling constraints during the interactions of heterogeneous and autonomous agents participating in a problem solving task, but it is also grounded in a practical implementation.

Acknowledgements

I would like to thank all those people who have supported me throughout this PhD. First and foremost, my deepest gratitude to both of my supervisors, Dr. David Robertson and Dr. Chris Walton, for all their time, support, friendship and intellectual inspirations. They have been the best supervisors both in wisdom and temperament, and this thesis would not have been possible without their help and support.

To all my fellow research students, especially to the past and present members of the Room 4.15, Appleton Tower; Adam Baker, Paolo Besana, Thomas French, Li Guo and Jarred McGinnis. Your kindness, friendship, hospitality and enlightening discussions proved invaluable.

I would also like to thank the Malaysian community in Edinburgh, especially the people of 32/10 Sinclair Place (Yr. 2003-2004), 5/8 Westfield Court (Yr. 2004-2006) and 5/11 Westfield Court (Yr. 2006-2007). I have been lucky in having great and wonderful friends, who have individually helped me in one way or another throughout the years.

I am also indebted to the Universiti Teknologi PETRONAS, Malaysia, which funded my research. Without its financial aid, this thesis would not be possible.

A special thank you to my family, especially my parents, Tn. Hj. Hassan Abdullah and Pn. Hjh. Fatimah Khamis, for the continual support and encouragement that I have received throughout my education. It is my family who is most responsible for inspiring me to strive for the goal to which this thesis is a testament.

Last but not least, my heartfelt thanks to a great friend, my wife and soul-mate, Zahiraniza Mustaffa. Her unfailing moral support in times of difficulties, understanding, and counsel have been invaluable in contributing to the successful completion of this thesis.

I sincerely thank all of you.

Declarations

I declare that this thesis was composed of myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Mohd Fadzil Hassan*)

Publications

Throughout the course of this degree, I have taken the advantage of immeasurably helpful input provided by the peer review system. Most of the ideas developed in this thesis have been presented or published for the following workshops and journals.

- Hassan, M.F., and Robertson, D. (2004). Constraint relaxation to reduce brittleness of distributed agent protocols. In Proceedings of the *Coordination in Emergent Agent Societies Workshop (CEAS' 04)*, held in conjunction with the *16th European Conference on Artificial Intelligence (ECAI' 04)*, Valencia, Spain.

- Hassan, M.F., Robertson, D., and Walton, C. (2005). Addressing constraint failures in agent interaction protocol. In Proceedings of the *Eight Pacific-Rim International Workshop on Multi-Agents (PRIMA' 05)*, Kuala Lumpur, Malaysia. To appear in the Lecture Notes in Artificial Intelligence series (LNAI), Lukose, D., and Shi, Z. (Eds.), vol. 4078, Springer Verlag.

- Robertson, D., Barker, A., Besana, P., Bundy, A., Chen-Burger, Y.H., Dupplaw, D., Giunchiglia, F., van Harmelen, F., Hassan, F., Kotoulas, S., Lambert, D., Li, G., McGinnis, J., McNeill, F., Osman, N., de Pinninck, A.D., Siebes, R., Sierra, C., and Walton, C. Models of interaction as a grounding for peer to peer knowledge sharing. To appear in the Advances in Web Semantics, Chang, E., Dillon, T., Meersman, R., and Sycara, K. (Eds.), vol. 1, LNCS-IFIP, Springer Verlag.

Table of Contents

1. Introduction **1**

1.1 Multi-Agent Systems 1

1.2 Coordination in MAS 4

1.3 Agent Interaction for Distributed Problem Solving 6

1.4 Scope, Motivation, and Aims 10

1.5 Thesis Outline 13

2. Related Work **15**

2.1 Agent Interaction 15

2.1.1 Approaches to Agent Interaction 16

2.1.2 Communication in Multi-Agent Systems 17

2.2 Objective-based Approaches for Agent Interaction 20

2.2.1 Electronic Institutions 20

2.2.2 Lightweight Coordination Calculus 24

2.3 Distributed Problem Solving Environment 34

2.4 Over-Constrained Problems 38

2.5 Partial Constraint Satisfaction Problems 42

2.6 Chapter Summary 46

3. Interaction Formalisation and Over-Constrained Problems **48**

3.1 Customer/Vendor Scenario 48

3.2 Sources of Brittleness in Interaction Protocols 58

3.3 Addressing Brittleness via Constraint Relaxation 66

3.4 Overview of Constraint Relaxation Approach 68

3.5 Chapter Summary 71

4. Protocol Specification **72**
4.1 Application of Distributed Partial CSP for Addressing Over-Constrained Problem 74
4.1.1 A Metric for Solution Subset Distance 76
4.1.2 Finding a Solvable MAP 81
4.1.3 Global Distance Computation 84
4.1.4 Constraint Relaxation 87
4.2 Algorithms for Finding Relaxed Problems that Achieve Solvable State with Minimal Distance 89
4.3 Implementing Constraint Relaxation Approach in LCC 95
4.4 Chapter Summary 104

5. Implementation and Working Example **105**
5.1 Implementation 105
5.2 Working Example 108
5.3 Chapter Summary 118

6. Evaluation **119**
6.1 Measures Used 119
6.2 Experimental Test Bed 122
6.2.1 Problem Generation Phase 123
6.2.2 Distributed Constraint Relaxation Phase 127
6.3 Experimental Results 128
6.4 Analysis of Results 131
6.4.1 Macro-level Analysis 132
6.4.2 Micro-level Analysis 136
6.5 Chapter Summary 141

7. Conclusions and Future Works **142**
7.1 Conclusions 142
7.2 Future Works 144
7.2.1 Employing Constraint Relaxation Strategies 144
7.2.2 Utilising Different Distance Metrics 144
7.2.3 Handling of Non-Crisp Constraints 145
7.2.4 Evaluation Based on Real-Life Applications 146

Appendix A. Prolog Code **147**
A.1 interface.pl 147
A.2 expansion_engine.pl 150
A.3 constraint_handling.pl 155

Bibliography **159**

List of Figures

Figure 1.1: Factors in a program's decision-making process 2
Figure 1.2: Cooperation topology 2
Figure 1.3: Categorisation of common dependencies among activities 4
Figure 1.4: Problem solving stages 7
Figure 1.5: Possible sub-problems of interacting agents 9
Figure 1.6: Conceptual model of agent interaction 9

Figure 2.1: A generic communication stack for agent interaction 18
Figure 2.2: Example of El and scene 23
Figure 2.3: Syntax of LCC protocol language 26
Figure 2.4: Diagrammatical view of LCC operators 26
Figure 2.5: Example of LCC protocol 27
Figure 2.6: Basic architecture of agent interactions in LCC 28
Figure 2.7: Message-passing in LCC 30
Figure 2.8: Variable interdependency in distributed problem solving 38
Figure 2.9: Example of an over-constrained problem 39
Figure 2.10: The problem space of partial constraint satisfaction 43
Figure 2.11: Example of partial CSP application to solve an over-constrained problem. 47

Figure 3.1: Roles and interaction diagram 49
Figure 3.2: Enacting distributed constraint solving interaction in LCC 58
Figure 3.3: Bilateral problem solving process 58
Figure 3.4: Conceptual overview of constraint graphs and solution lists involved in agents' interactions 61
Figure 3.5: Partially expanded interaction protocol clauses of the customer agent 64
Figure 3.6: Partially expanded interaction protocol clauses of the vendor agent 64
Figure 3.7: Interaction graphs for customer and vendor 65
Figure 3.8: Formal model of constraint relaxation interactions 69

Figure 4.1: Problem solving stages and constraint relaxation task 73
Figure 4.2: Possible relationships between the solution sets of the original(S) and the relaxed (S') problems 77
Figure 4.3: Equations for distance computation 78
Figure 4.4: The value of 'd' derived from the solution sets of the original and relaxed problems 79
Figure 4.5: Necessary bound for restraining the relaxed problems generated by agents . 80
Figure 4.6: The search for a solvable MAP state 83
Figure 4.7: Algorithm for constraint relaxation (i) 91
Figure 4.8: Algorithm for constraint relaxation (ii) 92
Figure 4.9: Algorithm for constraint relaxation (iii) 93
Figure 4.10: Algorithm for constraint relaxation (iv) 94
Figure 4.11: Interaction between agent roles 97
Figure 4.12: Encoding of constraint relaxation as a LCC protocol (i) 101
Figure 4.13: Encoding of constraint relaxation as a LCC protocol (ii) 102
Figure 4.14: Encoding of constraint relaxation as a LCC protocol (iii) 103
Figure 4.15: Encoding of constraint relaxation as a LCC protocol (vi) 104

Figure 5.1: Architecture for distributed constraint relaxation interactions 106
Figure 5.2: Problem space of the customer agent 110
Figure 5.3: Problem space of the vendor agent 111
Figure 5.4: Flow of inter-agent interactions 114
Figure 5.5: Selection of agents' CSPs during constraint relaxation computations 117

Figure 6.1: A complete relaxation cycle 121
Figure 6.2: Number of combination of r values from n possible values 124
Figure 6.3: The size of combinations for different values of r 125
Figure 6.4: Relaxation cycles for over-constrained MAPs with 25% compatibility level of local constraints. 133
Figure 6.5: Relaxation cycles for over-constrained MAPs with 50% compatibility level of local constraints. 133

Figure 6.6: Relationship between the parameters and number of relaxation cycles obtained 135
Figure 6.7: Comparison between 25% and 50% compatibility levels of local constraints– Over-constrained problem with a domain size of 4 139
Figure 6.8: Comparison between 25% and 50% compatibility levels of local constraints– Over-constrained problem with a domain size of 5 140
Figure 6.9: Comparison between 25% and 50% compatibility levels of local constraints– Over-constrained problem with a domain size of 6 140
Figure 6.10: Comparison between 25% and 50% compatibility levels of local constraints –Over-constrained problem with a domain size of 7 141

List of Tables

Table 2.1: Rewrite rules for expansion of a protocol clause 33

Table 3.1: LCC protocol for the given scenario 50
Table 3.2: Sequence of message passing 62

Table 4.1: Distance metric computation for achieving a MAP solvable state 85
Table 4.2: Distance metric computation for achieving a MAP solvable state 85
Table 4.3: Distance metric computation for achieving a MAP solvable state 86

Table 6.1: Protocol's performance against over-constrained MAPs with 25% compatibility level of local constraints 129
Table 6.2: Protocol's performance against over-constrained MAPs with 50% compatibility level of local constraints 130

List of Definitions

Definition 2.1: Constraint Satisfaction Problem (CSP) 36
Definition 2.2: Distributed Constraint Satisfaction Problem (DCSP) 36
Definition 2.3: Multi-Agent Agreement Problem (MAP) 37
Definition 2.4: partial Constraint Satisfaction Problem 43
Definition 2.5: distributed partial Constraint Satisfaction Problem 45

Chapter 1

Introduction

The work described in the thesis brings together established works from two separate research disciplines; the constraint satisfaction and multi-agent system research fields. It specifically demonstrates on how an established technique for addressing over-constrained problem within the constraint satisfaction research field can be specified as a distributed agent protocol. This allows for a more flexible interactions among agents involved in a distributed problem solving task. The integration of constraint satisfaction techniques in multi-agent systems is a growing research area [Calisti and Neagu, '04], and this work enriches this expanding research area in the following two general aspects.

1. For the constraint satisfaction research field, it makes the available techniques to address over-constrained problem relevant for the peer-to-peer agent environment.
2. For the multi-agent system research field, particularly the distributed agent protocol, it addresses the brittleness problem commonly faced by problem solving agents during their interactions for finding a solution.

This thesis begins by giving a general overview on multi-agent systems, coordination in multi-agent systems, followed by a discussion on how agent interactions provide the means to coordinate agents in a distributed problem solving environment. Next, a discussion on the motivation and aim of the research work will be provided. A description of the thesis' remaining chapters will conclude this introductory chapter.

1.1 Multi-Agent Systems

A software entity is generally accepted and recognised as an "agent" if it can exhibit an autonomous feature, which means it is able to perform an independent computational activity and interacts with its surrounding environment [Wooldridge and Jennings, '95].

Its behaviour is directed *not only* by its own experience (stored knowledge), but by its ability to evaluate and combine this with knowledge about the current situation and the environment, along with any other pertinent information available to produce an autonomous and deliberative decision making process [Chalmers, '04], as illustrated in figure 1.1.

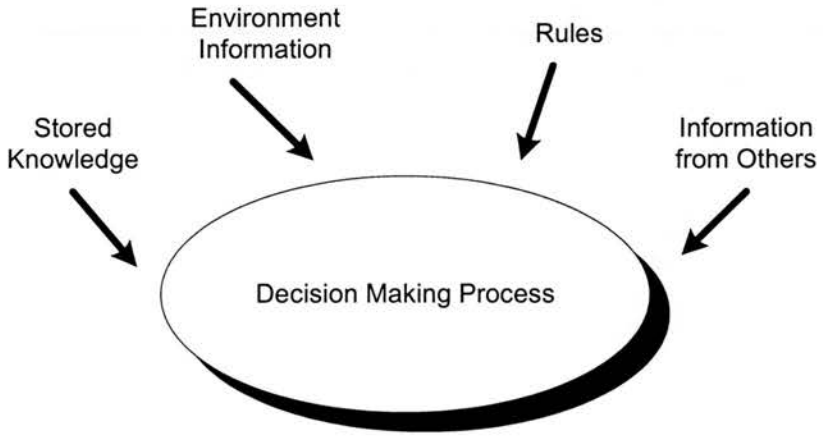


Figure 1.1: Factors in a program's decision-making process

An important aspect of the agent-based approach is the principle that agents (like humans) can function more effectively in groups that are characterised by cooperation and division of labour [Chaib-Draa and Dignum, '02]. In fact, cooperation is often presented as one of the key concepts which differentiates multi-agent systems (MAS) from other related disciplines such as distributed computing, object-oriented systems, and expert systems. The broad view description of cooperation within the context of MAS can be illustrated using the topology provided in figure 1.2 [Doran et al., '97; Franklin and Graesser, '97].

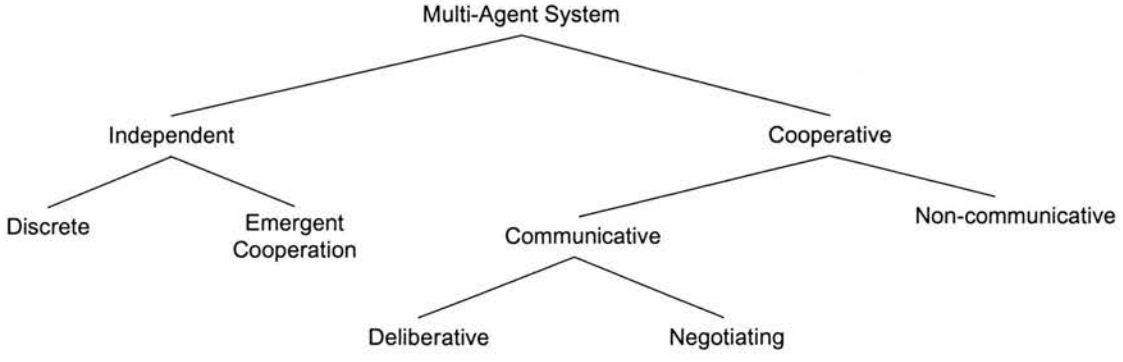


Figure 1.2: Cooperation topology

A MAS is *independent* if each agent pursues its own agenda independently of the others. A system is *discrete* if it is independent, and if the agendas of the agents bear no relation to one another. A system can be described as having *emergent cooperation* behaviour if from an observer's viewpoint, the agents appear to be working together, but from the agent's viewpoint they are not. They are simply carrying out their own individual behaviour.

The opposite of independent systems are *cooperative* systems, in which the agendas of the agents include cooperating with other agents in some way. Such cooperation can be either *communicative* in that the agents communicate with each other in order to cooperate or it can be *non-communicative*. For the non-communicative form, agents coordinate their cooperative activities by each agent observing and reacting to the behaviour of the others. On the other hand, communicative cooperation can be in at least two forms – *deliberative* or *negotiating*. In deliberative systems, agents jointly plan their actions in order to cooperate with each other. Negotiating systems are similar to deliberative systems, except that they have an added element of competition.

A more precise and constrained definition emphasised that cooperation occurs when the actions of each agent satisfy at least one of the following conditions [Doran et al., '97]:

1. The agents have a (possibly implicit) goal in common (which no agent could achieve in isolation) and their actions aim at achieving that goal.
2. The agents perform actions which enable them to achieve not only their own goals, but also the goals of other agents.

Following these definitions, the scope of the research work reported in the thesis is primarily concerned on MAS which can be viewed as a loosely coupled network of problem solvers that interact to solve problems that are beyond the individual capabilities or knowledge of each problem solver. The problem solvers are autonomous and can be heterogeneous in nature [Sycara, '98].

One of the biggest concerns in any distributed reasoning system is how the action of the individual agents can be coordinated so that they work together effectively [Rich and

Knight, '91]. Therefore, in the next section, a general overview of coordination in MAS is given.

1.2 Coordination in MAS

A general definition of coordination is provided by the coordination theory introduced in [Malone and Crawston, '94]. In this theory, coordination is viewed as a process of managing dependencies between activities, and the categorisation of these dependencies is provided in figure 1.3. According to this theory, autonomous entities need to coordinate their actions in order to manage the dependencies that exist between these activities.

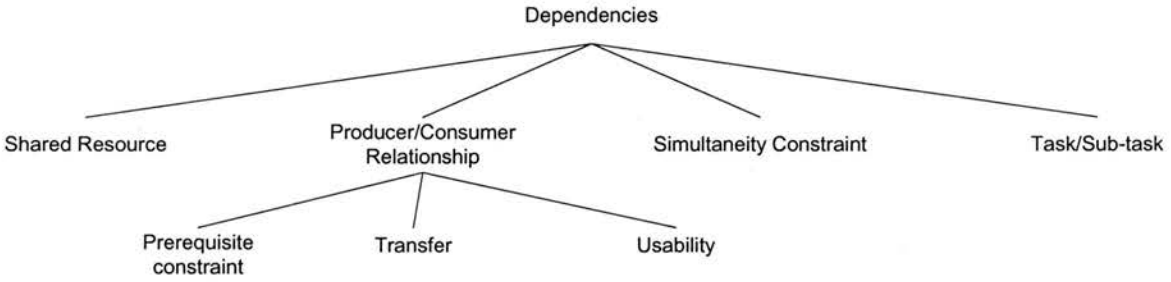


Figure 1.3: Categorisation of common dependencies among activities

In a definition that focuses specifically on MAS, coordination is viewed as a process in which agents engage in order to ensure a community of individual agents acts in a coherent manner. Coherence means that the agents' actions are consistent with each other. In other words, coherence refers to how well a system of agents behaves as a unit, and these agents need to be coordinated for the reasons described in [Nwana et al., '96]:

- Preventing anarchy or chaos – coordination is necessary or desirable because, with decentralisation in agent-based systems, anarchy can set in easily. Agents no longer possess a global view of the entire group to which they belong. Consequently, agents only have local views, goals and knowledge, which may conflict with others. They can enter into all sorts of arrangements with other agents or agencies. Like in any society, such haphazard arrangements are prone to anarchy; to achieve common

goals, which are a major reason for having multiple agents in the first place, a group of agents need to be coordinated

- Meeting global constraints – there usually exist global constraints which a group of agents must satisfy if they are to be deemed successful
- Distributed expertise, resources or information – agents may have different capabilities and specialised knowledge
- Dependencies between agents' actions – agents' goals are frequently interdependent
- Efficiency – even when individuals can function independently, thereby obviating the need for coordination, information discovered by one agent can be of sufficient use to another agent that both agents can solve the problem faster

Coordination among agents is accomplished through social interactions, one of the fundamental features of MAS. These social interactions are enacted through a variety of interaction protocols, here regarded as the public rules or norms for communications of the participants of a group when carrying out some social encounter. In this context, the protocol ensures that all participants following it can expect certain responses from others and can coordinate meaningfully towards a goal [Paurobally et al., '03].

1.3 Agent Interaction for Distributed Problem Solving

As described in [Faratin and Klein, '01], collaborative problem solving task which uses the MAS approach is generally composed of the following three stages: pre-interaction, interaction and post-interaction. These stages are illustrated using a simplified diagram in figure 1.4, which involves two agents (agent a and b). Boxes, ovals and links represent processes involved in the decision making, data, and information flow respectively.

The local problem of the agent is defined in the pre-interaction phase. This stage of the collaborative activity, which is non-interactive, can be informally described as the stage where the agent "gets to know itself and what it wants". At this stage the agent attempts to not only define its local problem, but it may also attempt to solve the problem independently of interactions with other agents. A local solution is a locally consistent assignment of values to a set of variables that satisfy some set of domain constraints. Let j ($j \in \{a,b\}$) represent an agent. Let $I^j = \left\{ i_1^j,...,i_n^j \right\}$ represent the local n-dimensional variables, or issues, of agent j. Domain constraints are local/endogenous restrictions on the local decision making, which include a minimal unary constraint of the domain, or reservation value for each of the variable. Another possible constraint are the binary/*n*-nary dependencies between the variables. In this research, we restrict ourselves to the problems in which both the set and ontology of the variables are shared among the agents, that is $I^a=I^b$.

Once agents have a consistent assignment of values to each of their local variables, they enter the next stage of the collaborative activity which is interaction. This stage is specifically concerned on the modification and checking of consistency of the joint set of constraints. Conflicting preferences (or interaction constraints) make the achievement of a mutually agreed set of values for a variable difficult to achieve. The post-interaction stage is essentially a commitment problem where mutual agreement on the values of the set of variables achieved during interaction stage must be honoured.

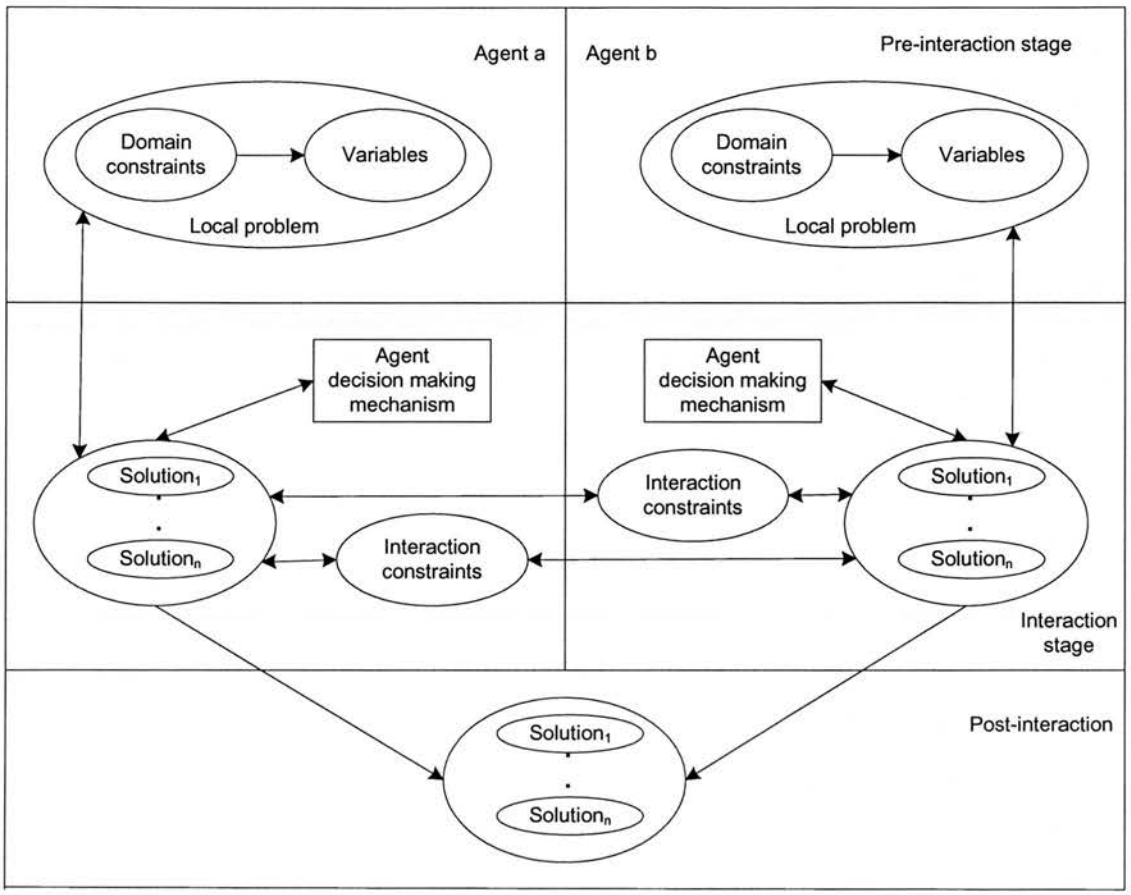


Figure 1.4: Problem solving stages

Depending on the kind of sub-problem interdependencies, the interaction among agents in a multi-agent system for a distributed problem solving task can be complex, often requiring a multi-step dialogue. This interaction can be achieved through a protocol that provides not only the communication of agents, but also the creation and destruction of agents (i.e. agents entering/leaving a MAS), the spatial distribution of agents, as well as synchronisation and distribution of actions over time [Bocchi and Ciancarini, '03]. It generally involves two important elements – the subjects whose activities need to be coordinated (i.e. agents) and the entities between which dependencies arise (i.e. objects of coordination), namely sub-problems handled by the individual agents [Omicini and Ossowski, '03].

The specification of the involved elements and the relationship that exist between them are generally mediated and represented by the notion of *role*. When assuming a role, an agent is in charge of the corresponding task or action, and is entitled to all the

authorisations and permissions (and limitations as well) pertaining to its role. This can be viewed as social norm constraints imposed on the agents upon assuming the roles specified in the protocol. The state of the agent interactions is then reflected on the ways these constraints are mutually and individually satisfied by the interacting agents.

The following is a short but (by current standards) complex scenario that deals with the purchasing and configuration of a computer between the customer and vendor agents, which is borrowed from [Robertson, '04c], to describe agent interactions for a distributed problem solving task:

> An internet-based agent acting on behalf of a customer wants to buy a computer but doesn't know how to interact with other agents to achieve this, so it contacts a service broker. The broker supplies the customer agent with the necessary interaction information. The customer agent then has a dialogue with the given computer vendor in which the various configuration options and pricing constraints are reconciled before a purchase is finally made.

To simplify the discussion, it is assumed that the interaction between the vendor and customer agents is concerned on only four abstract attributes namely V_1, V_2, V_3 and V_4. Figure 1.5 provides an abstraction of a generic description of the possible problems of the agents, which can range from a loosely constrained sub-problem (i.e. P_1) where each variable is independent of each other and solely constrained by the assigned domain values to a densely constrained sub-problems (i.e. P_2 and P_3) where variables are interdependent. P_1 is formally known as a unary constrained problem while P_2 and P_3 can be regarded as n-nary constrained problems, with different degrees of hardness [Tsang, '93].

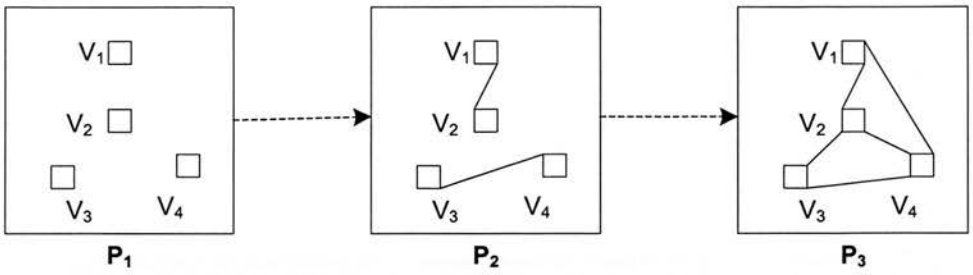


Figure 1.5: Possible sub-problems of interacting agents

The means of communicating and coordinating the problem solving efforts given the distinct sub-problems of the customer and vendor agents can be provided through an interaction model, as abstractly described in figure 1.6.

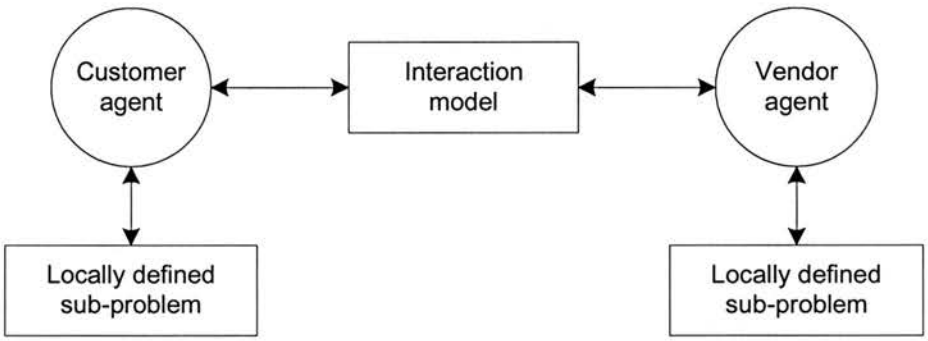


Figure 1.6: Conceptual model of agent interaction

The interaction model provides roles (i.e. customer and vendor) that could be assumed by the interacting agents for reconciling their distinct sub-problems in finding mutually acceptable values for all the four variables. The interactive states of the agents communicating through this model are dependent on the satisfiability of the constraints associated with the variables of the problem to be solved. The computation performed on an interaction model might involve the execution of the roles contained in the model across different machines or agents, therefore satisfaction of constraints by an agent associated with a particular role in an interaction model is done in ignorance of constraints imposed by other agents in the interaction. Hence, for a successful termination of the interaction model in coordinating the agents to achieve the intended objective of finding an agreeable solution, we require all constraints associated with the agents' roles to be solvable. For instance, within the given scenario, the agents are in conflict if no

compatibility is found between the corresponding variables values defined by the interacting agents. This conflict may lead to a failure in the reconciliation process, preventing the agents' progressions in their respective prescribed roles of the interaction model for achieving a solvable state. This inconsistent local view of interacting agents, which causes interaction failure, can be perceived as an over-constrained problem.

As such, interaction models are considered brittle, in a sense that the constraints imposed on the roles contained in the models must either succeed or fail, and if they fail the entire models may fail to achieve the objective of adequately resolving the interdependence among the agents' sub-problems. Consequently, protocol failure can cause the interaction between agents to break prematurely, even though the interacting agents could in principle reach an agreement.

This problem will be re-visited again in chapter 3, illustrated in detail via an interaction model, formalised and executed using a particular distributed interaction protocol language called the Lightweight Coordination Calculus (LCC).

1.4 Scope, Motivation, and Aims

In many constraint satisfaction research works, an extensive use of the terms 'agent' and 'agent interaction' can be found [Yokoo, '93; Yokoo et al., '98; Yokoo, '01]. However, it should be clearly noted that these two terms have been used in the constraint satisfaction and MAS worlds with slightly different meanings [Calisti and Neagu, '04].

In the constraint satisfaction research field, an agent is a computational entity acting as a decision maker following pre-defined coordination mechanisms (i.e. constraint solving mechanisms or algorithms) and sharing an implicit common representation of the world with other agents (i.e. no explicit use of structured communication stack and ontologies). In contrast, from a MAS perspective, an agent is autonomously deciding whether or not to follow specific coordination mechanisms and can communicate with other agents by means of structured semantic-grounded exchange of messages.

Within the constraint satisfaction world, the fundamental issue is on how to obtain a consistent assignment of values to a set of variables maintained by distinct agents within a distributed environment, with a very little emphasis put on the communication model

employed by the agents. The requirement to communicate is mainly driven by the algorithm used for finding a solution to a given constrained problem. In a way, the definition of agent in the constraint satisfaction world is very much equivalent to the 'weak agency' of MAS. Given the slightly distinct definitions of the terms 'agent' and 'agent interaction' as specified in the constraint satisfaction and MAS communities respectively, any mentioned of these terms in this thesis is implicitly assumed to be the definitions provided by the latter unless it is duly noted otherwise.

The work presented in this thesis considers a stronger notion of agency. It specifically focuses on how an explicit interaction model used in synchronising the message-passing behaviour of heterogeneous problem solving agents can be affected by failure of any of the agents involved in the interaction process to satisfy the constraints imposed on the individual agent roles engineered within the model. Given that the participating agents are capable of relaxing their individual constraints to accommodate the constraints of others in order to reach a common goal, the main issue this work tries to address is how agents could communicate and coordinate the constraint relaxation process. This can be achieved by providing agents with some safe envelope of constraint bounds across the interaction for reconciling their sub-problem differences. The interaction mechanism is obtained by re-interpreting a technique borrowed from the constraint community, deployed at the interaction protocol level.

It is not the aim of this research to provide any new advance in the already mature fields of constraint satisfaction. Instead, it attempts to bridge the gap between the worlds of constraint satisfaction and MAS by promoting the use of the techniques established by the former to solve a class of a distributed interaction problem faced by the latter. It is the aim of this research to extend the capability of conceptual and theoretical techniques for addressing over-constrained problems within the constraint satisfaction field by reinterpreting these techniques from the distributed agent protocol perspective. The study is concerned with how existing approaches used to address over-constrained problems in constraint satisfaction field can be integrated and adapted within a distributed interaction protocol framework to have a more flexible means of constraint handling during agent interactions. For this purpose, we focus on a particular interaction protocol language called the Lightweight Coordination Calculus (LCC).

As described in [Sycara, '98], the use of constraint satisfaction techniques in MAS is not new as they have been utilised either as a part of the agents' problem solving apparatus or coordination formalisms as reported in [Macho-Gonzales et al., '00; Aldea et al., '01; Meisels and Kaplansky, '02]. However, in these works, the focus is strictly on the conventional formalisms of constraint satisfaction which require all constraints to be satisfied and do not address over-constrained problems. The few approaches that do attempt to integrate the currently available constraint satisfaction techniques for over-constrained problem with MAS include that of [Luo et al., '03], which proposed a fuzzy constraint-based model for bilateral multi-issue negotiations in MAS. The work is applied to an accommodation-renting scenario involving a negotiation between a prospective tenant and a letting agency. The model is directly engineered as part of the internal functionality of the interacting agents.

This thesis, on the other hand, considers the problem of integrating a particular constraint satisfaction technique for solving an over-constrained problem (i.e. distributed partial Constraint Satisfaction Problem) as part of the constraint-handling feature of the distributed interaction protocol system (i.e. LCC). This is a novel way of providing a more flexible approach for handling constraints during the interactions of heterogeneous and autonomous agents participating in a distributed problem solving task. The proposed approach is not specifically engineered as part of the agency, and its deployment and execution does not rely on any centralised mechanism. In this way, the brittleness of agent interaction due to the conflicting constraints imposed by the individual agents can be addressed by the agents themselves without any third-party intervention.

The use of a third-party mediator for coordinating the relaxation of an over-constrained situation might be acceptable if confidentiality is not the main concern; so it is acceptable for agents to reveal their internal goals to the third parties. However, in some domains (e.g. customer-vendor bargaining interaction), it is not practical for this private information to be completely revealed, as it might jeopardise the agents' individual strategies for obtaining an optimal outcome from the interaction process. As emphasised in [Pruitt, '81; Rosenschein and Zlotkin, '94], it is important that the agents minimise the amount of information they reveal about their preferences since any such revelation can weaken their bargaining position. Another reason why the agents need to

minimise such information revelation is that humans, depending on the nature of interaction, are not always willing to completely disclose private information while interacting with each other [Heiskanen et al., '01]. Thus, if we want interacting agents to actually represent humans, they must follow the same broad tenet. In addition, heavy dependency on a third-party agent to resolve any arising conflicts might lead to bottlenecks. It is, hence, more advantageous for the constraint relaxation approach to be managed directly by the involved agents themselves.

1.5 Thesis Outline

- Chapter 2 reviews the literature on approaches to agent interaction, and the interaction protocol language used in this research (i.e. LCC). It also includes a description on the distributed problem solving environment and approaches for handling over-constrained problems within the constraint satisfaction research field.

- Chapter 3 provides a discussion on the interaction model, formalised and executed using LCC for a particular scenario. Using an example, brittleness of the interaction model due to an over-constrained problem is described.

- Chapter 4 provides a detailed description of realising a distributed partial CSP as an LCC protocol.

- Chapter 5 provides a discussion on the implementation of our approach and detailed execution of the constraint relaxation protocol using the scenario of chapter 3.

- Chapter 6 provides a description on the test bed used in evaluating the protocol. This chapter also provides an analysis on the results obtained from the evaluation.

- Chapter 7 concludes with a summary and a discussion of future avenues for research on this topic.

Chapter 2

Related Work

This chapter provides a detailed review on two areas deemed important to the research reported in this thesis; agent interaction and distributed, over-constrained, constraint satisfaction problems. For the former, this include a review on approaches to agent interaction, and the interaction protocol language used in this research (i.e. LCC). A description on the distributed problem solving environment and approaches for handling over-constrained problems within the constraint satisfaction research field are provided for the latter.

2.1 Agent Interaction

Agents populating a MAS can be mainly classified as either benevolent (cooperative) or self-interested [Lesser, '99]. Cooperative agents work toward achieving some common goals, whereas self-interested agents have distinct goals but may interact to advance their own goals. In the latter case, self-interested agents may, by exchanging favours, coordinate with other agents in order to get those agents to perform activities that assist in the achievement of their own objectives.

In both classifications, the need for interaction between agents is absolutely essential because it enables the MAS to exist. If agents are not able to interact with one another, no global behaviour in the MAS is possible, and hence the fundamental benefits of using a MAS approach could not be fully gained. Agent interaction becomes a critical issue in MAS as it allows interdependency that exists between agents to be coordinated, in order for the agents' overall goals to be achieved [Schumacher and Ossowski, '06]. Given this consideration, computational agents require ordered and structured interactions [Bond and Gasser, '88]. Such structuring is needed because in the absence of any normative rules of public behaviour, interactions lead to chaotic dynamics where agents can send messages that cannot be understood or the message is inappropriate

given the history of the current interaction [Faratin, '00]. Therefore, in this research, we are interested in the interactive aspect of agents particularly for MAS-based distributed problem solving systems. This section then provides an introductory overview of aspect fundamental to the focus of the research work reported in this thesis: approaches to agent interaction and communication in MAS. These are followed by an overview on two objective-based approaches to agent interaction; Electronic Institution (EI) and Lightweight Coordination Calculus (LCC), in the subsequent section.

2.1.1 Approaches to Agent Interaction

The many diverse approaches to the multi-agent interaction can be categorised in two main classes – the subjective and objective approaches [Omicini and Ossowski, '03].

In the subjective approach, interaction is encapsulated as part of the intra-agent components. Interactions are possible through the specification and development of agent languages and architectures, closely integrated with the agent's internal structure. With this approach, each agent is assumed to have an understanding of the basic communication elements to enable it to establish interactions. Given its state, it is expected to infer the appropriate interaction action. The global behaviour of the system emerges from all the individual interactive decisions made by each agent. This allows for the greatest amount of autonomy for individual agents but at the risk of disorder or break down of the system as the complexity of the interactions increase [McGinnis, '06]. The subjective approach is widely used and it includes mentalistic or Belief-Desire-Intention (BDI) model [Bratman, '87] of agent interactions based on the speech act theory of [Austin, '62; Searle, '69].

As interactions may occur between similar or different agents within the same system or across heterogeneous environments, sole dependency on the subjective approach for coordinating agent interactions proved to be inadequate and led to a number of problems, including the semantic verification problem [Wooldridge, '00]. This gives rise to the objective approach which argues that several aspects of multi-agent systems that conceptually do not belong to agents themselves should not be assigned to, or hosted inside agents. Examples include infrastructure for communication and coordination, the

topology of a spatial domain, and support for the action model [Schumacher and Ossowski, '06]. In the objective approach, a MAS is not simply considered as a sum of individuals. Instead, a MAS is perceived as a society of agents where a collective social behaviour is likely to emerge. This society defines not only the world where agents live, but also the laws that permeate the interaction space or the communication media that enable agent interactions [Bocchi and Ciancarini, '03]. The society has norms and traditions. For agents to participate in a MAS and thus participate in the society, it is the responsibility of each individual agent's engineer to design his/her agents to follow the rules of the society. The consequence of this is a more reliable agent interaction. It is also more scaleable due to the ability to know the global state of the MAS as interaction activities are specified by the society. This comes at the cost of autonomy. Agents are not completely free to explore the interaction space, that is the set of all possible meaningful sequences of messages given an agent communication language. Agents can only converse by following the sequences allowed by the society [McGinnis, '06]. This objective approach necessitates a clear identification of the interaction setup in a MAS, which naturally calls for a separation between the design of each individual agent and the design of their interactions [Schumacher and Ossowski, '06]. Further details with regards to the objective-based approaches to agent interactions including the Lightweight Coordination Calculus (LCC), an interaction protocol language used in the research work, are described in section 2.2.

2.1.2 Communication in Multi-Agent Systems

Regardless of the high-level approaches used to mediate agent interactions, there exist some communicative aspects that need to be shared among agents to ensure that a proper and smooth interaction can take place. These communicative aspects can be described using a generic communication stack [Calisti, '02], which is composed of low-level data-transport level and abstract components used at the higher communication level, as illustrated in figure 2.1.

At the lowest level, a transport layer consists of basic building blocks responsible for transparently routing and delivering agent messages to the final intended recipient(s).

On top of the transport infrastructure, agents interoperate by parsing and interpreting messages in the context of on-going conversation, achieved through components of the communication layer. Brief descriptions of the abstract components of the communication layer are as follows:

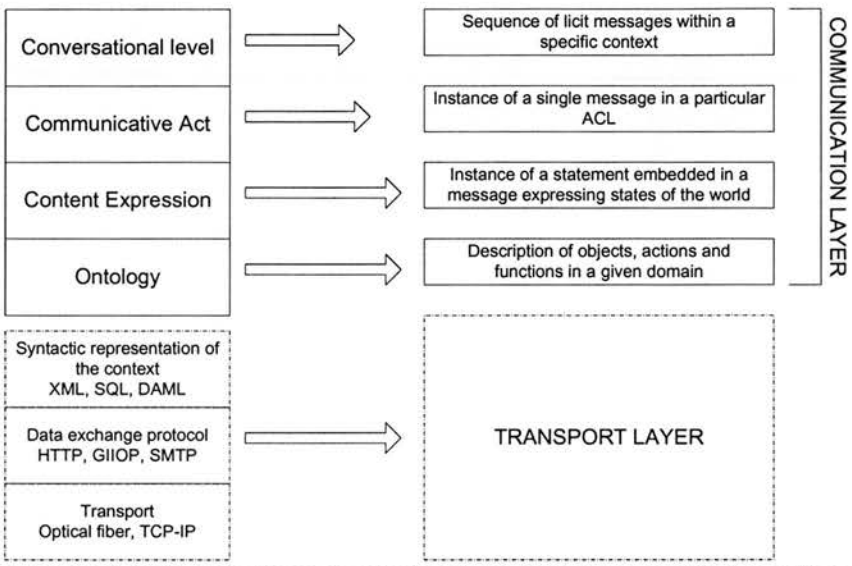


Figure 2.1: A generic communication stack for agent interaction

Interaction Protocols for Agent Conversations. A conversation is regarded as an interaction protocol instance or occurrence [Jouvin and Hassas, '02]. Conversation can be defined as a succession of messages (communicative acts) exchange between two or more agents following a well defined interaction pattern called protocol. An interaction protocol defines several agent roles, which defines the set of responsibilities of one interlocutor participating in the conversation. Several roles may be impersonated by a single agent. Conversations are the instantiation of interaction protocols in actual systems, and are by themselves a basic organisational construct, in that they define a relationship between interlocutors, and their roles.

Interaction protocols are used to specify the set of allowed message types (i.e. performatives), message contents and the correct order of messages during the conversations between agents [Odell et al., '00; Lind, '01; Odell et al., '03], and they can become the basis for agent negotiation and cooperation [Chen and Sadaoui, '03]. Interaction protocols can force agents to act correctly in predictable ways. Using the protocols, the autonomous behaviours of agents can be predictable because agents are

obliged to obey some rules. The interaction protocols can range from complicated negotiation schemas to a simple request for a task to be performed. This layer that governs whole patterns of interactions, social norms, and communication within MAS is the ultimate concern of the research work reported in this thesis.

Agent Communication Language (ACL) and Content Expression. Once the valid sequence of possible communicative acts is known, it is necessary that agents parse and interpret every message they receive. This requires the adoption of a standard ACL and a knowledge representation language that have a precisely defined syntax and semantics. The ACL provides an agent with a set of performatives or locutions allowing it to communicate and express its intentions in accomplishing some task. These performatives or locutions are used to wrap the message content expressed in a knowledge representation language. So, the proposition "Reasonable temperature" can have a different meaning if it is expressed within a locution that is specified as a query, command or statement.

The first ACL to gain wide recognition is the Knowledge Query and Manipulation Language (KQML), which was proposed along with the Knowledge Interchange Format (KIF) as a means for knowledge sharing in the early 1990's [Finin et al., '94; Wooldridge, '02]. The development of KQML was an attempt to provide a set of performatives to capture the various propositional attitudes an agent might want to express, while KIF [Genesereth and Fikes, '92] focused on the representation of knowledge of a certain domain.

A number of limitations associated with the KQML have led to the development of FIPA-ACL [FIPA, '01]. FIPA-ACL offers the same functionality as KQML, but with improvements like the introduction of formal semantics.

Ontology Definition. The description of the world state that the agents are communicating about may contain references to *objects*, *actions* and *functions* (i.e. object models) in one or more domains. An ontology provides a vocabulary (class model) for representing and communicating domain-dependent knowledge, including a set of

relationships and properties that are valid for the elements identified by that vocabulary [Chnadrasekaran et al., '99].

The communication language, content language and the ontology must somehow be agreed by the participating agents, regardless of the approaches adopted in governing the agent interactions. For our work, this assumption is made because without it, little progress could be made.

2.2 Objective-based Approaches for Agent Interaction

The rapid evolvement of the field of agency gives rise to the development of a number of objective-based approaches to agent interaction. This section describes the existing literature in the field, focusing on two approaches namely Electronic Institution (EI) and Lightweight Coordination Calculus (LCC). The former is a prominent and popular technique for specifying and deploying agent interaction protocols in MAS while the latter has evolved due to dissatisfactions attributed to the shortcomings of the former. EI has a significant role in the development of LCC, a distributed protocol language that provides the foundation for the research work reported in this thesis. As such, the following sub-section is dedicated to provide a review on EI and its features before LCC is described in details in sub-section 2.2.2.

2.2.1 Electronic Institutions

The objective-based paradigm of agent interactions is largely typified by EI [Noriega, '97; Esteva et al., '00] . The underlying concept behind the framework is that human interactions are always guided by formal and informal conventions. Human interactions are never completely unconstrained; rather such notions as conventions, customs, etiquette, and laws control them. EI framework provides a means for controlling the interactions of agents in a MAS using formal institutions [Esteva et al., '01].

An EI is considered analogous to a theatre production. The agents that are coordinated by the institution are analogous to the actors, and each agent takes one or more roles in the institution. The interactions are articulated through the use of scenes in

which groups of agents directly interact. Within a scene, all the participating agents follow a single script which guides their interactions.

Though there exist a number of EI frameworks which vary in details, their basic principles are close to the ISLANDER [Esteva et al., '02]. The ISLANDER framework formally defines several core aspects of EIs. Central to ISLANDER are the formal definition of roles for agents, a shared dialogical framework, the division of the institution into a number of scenes and a performative structure which dictates, via a set of normative rules, the relationship between scenes.

The notion of role is central in the specification of EIs and each role defines a pattern of behaviour within the institution. A role can be defined as a finite set of actions, intended to represent the capabilities of the role. For instance, an agent assuming the buyer role is capable of submitting bids and an agent assuming the auctioneer role can offer goods at auction. In order to take part in an EI, an agent is obliged to adopt some role(s). Thereafter, an agent playing a given role must conform to the pattern of behaviour attached to that particular role. Therefore, all agents adopting similar roles are guaranteed to have the same rights, duties and opportunities.

In order to allow agents to successfully interact with other agents, the fundamental issue of having a common language and ontology must be addressed. This guarantees the interacting agents to have a shared vocabulary for communication as well as a common world-view with which to represent the world they are discussing. For this purpose, EI dictates that agents must share a dialogical framework when communicating. By sharing a dialogical framework, heterogeneous agents are capable to exchange knowledge and information with the other agents. The core of the dialogical framework includes an ontology, a content language, and a set of illocutions. The content language allows for the encoding of knowledge and information to be exchanged among agents using the vocabulary offered by the ontology, and this part makes up the inner language. The propositions generated using the inner language need to be embedded into an outer language, the communication language which expresses the intentions of the utterance by means of the illocutions, before being passed between the agents. The dialogical framework, which consists of the ontological elements, is essential for the specification of scenes.

All interaction between agents occurs within the context of scenes. A scene defines a generic pattern of interaction protocol between roles, expressed as the set of valid sequences of illocution that agents assuming the role can exchange. Any agent participating in a scene has to play one of its roles. A scene is specified as a directed graph where the nodes represent the different states of the interaction and the directed arcs connecting the nodes are labelled with the actions that make the scene state evolve. Each scene has a set of entrance and exit states, and agents participating in the scene must satisfy conditions associated with these states before they can enter or exit a scene.

As agents might be involved in a number of individual scenes, the relationship between these scenes needs to be properly formalised. The performative structure defines this network of scenes and their inter-relation with each other. It specifies how the agents depending on their role can move among different scenes, taking into account the relationship among the different scenes. The roles adopted by an agent and the actions performed by the agent upon assuming these roles create obligations and affect future actions available to the agents. These consequences can either limit or enlarge its subsequent possibilities for action, and provide a possible path for an agent within the performative structure. These are referred to as normative rules, and can be categorised as either *intra-scene* or *inter-scene*. Intra-scene dictates actions for each agent role within a scene, and inter-scene is concerned with the commitments which extend beyond a particular scene and into the performative structure [Esteva et al., '00; Esteva et al., '01].

In order to illustrate the concept of institution and scene, an example is provided in figure 2.2, which is borrowed from the work reported in [Walton and Robertson, '02]. Figure 2.2(a) provides an example of an institution designed for the diagnosis of breast cancer and one of the scenes for this institution is illustrated in figure 2.2(b). The institution consists of a number of inter-linked scenes. The rectangles represent scenes, and the inter-scene connectives represent the performative structure. The scene of figure 2.2(b) is intended to represent a patient(P) visiting a doctor(D) to obtain a diagnosis of breast cancer symptoms.

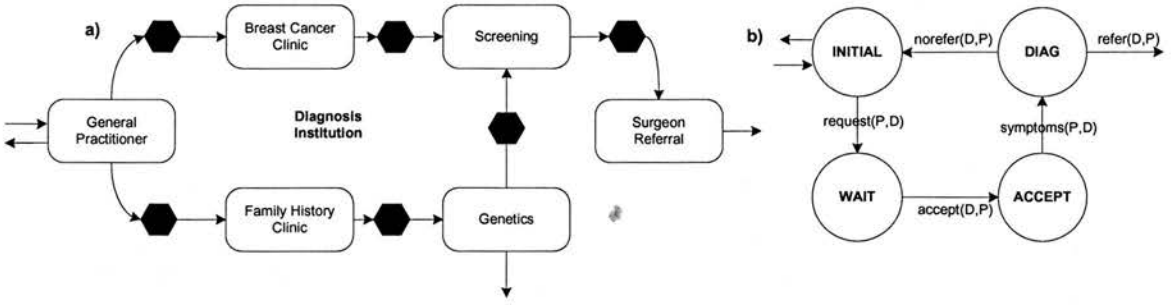


Figure 2.2: Example of EI and scene

The specification of EI as defined using the ISLANDER framework is executed using AMELI [Esteva et al., '04], the infrastructure and governing agent that mediates participating agent's interactions. Given an ISLANDER EI specification, AMELI ensures that agents participating in the institution adhere to all the specified norms. The innovative contribution of AMELI is its ability to implement any EI specification defined in ISLANDER regardless of domain.

Although not always described as such, EI is a form of protocol that is intended to be globally understood by the agents concerned. It relies on agents being aware of the current state of the institution, when and where they are expected to interact. A key issue with such a protocol, however, is how the global control is enforced in practice. The current enforcement technique (i.e. AMELI) relies on the use of administrative agents, or agent proxies to ensure the smooth running of the institution. It is through this central, coordinating agent (or "governor") that all messages associated with the institution are routed to. The governor can then enforce sequencing as necessary; prompt agents for appropriate messages; and generally keep the interaction coherent. The problem with this solution is that the agents are dependent on the governor to provide the necessary coordination for effectively interacting with each other. As argued in [Walton and Robertson, '02; Robertson, '04a], the use of governor undermines a key principle of agency–that each agent can operate autonomously–since governors remove part of that autonomy. In addition, the governor can become a bottleneck in agents' interactions if only a small number of governors are available to accommodate a sizeable number of interacting agents.

These limitations affect EI appropriateness for open heterogeneous MAS, and give rise to the development of distributed protocol approach that preserves agents' autonomy; does not rely on the governor, yet provides interaction coordination.

2.2.2 Lightweight Coordination Calculus

Distributed protocols are a new approach to multi-agent interaction. In the common practice, an agent's communicative model is developed by the individual engineers by interpreting any formal, graphical or natural language descriptions of a multi-agent system's interactions. The distributed protocol method, however, takes the view that the agents themselves can participate in a communication using a given interactive model if they are provided with the means to compute their parts in the interaction as specified in the model. The advantage of distributed protocols is that agents are not tied to a set of predefined protocols that their creator foresaw. A number of existing approaches for distributed protocols include [deSilva, '02; Freire and Botelho, '02], however as described in [McGinnis, '06], Lightweight Coordination Calculus (LCC) is considered more developed since it is readily available in an executable form and can be directly utilised for the work presented in this thesis. This does not necessary mean that our work is solely dependent on LCC. It is portable to any distributed protocol platform that has the same features as LCC, to be described in the remaining of this section, with very minimal adjustments.

The development of LCC is mainly driven by the dissatisfaction with the EI approach for agent interactions, especially the ISLANDER approach due to the described limitations. A detailed discussion concerning LCC specification and the means to compute interaction protocol terms within the LCC are given in the following two sub-sections.

2.2.2.1 LCC Syntax

LCC borrows the notion of role from agent systems that enforce social norms (e.g. EI) but reinterprets this in a formalism based on process calculus. The syntax of the protocol

language, taken from [Robertson, '03] is shown in figure 2.3. Social norms in LCC are expressed as message-passing behaviours associated with roles. In LCC, the interaction framework is composed of a set of clauses, each of which defines how a role in the interaction must be performed. Roles are described by the type of role and an identifier for the individual agent undertaking that role. The definition of performance of a role is constructed using combinations of the sequence operator ('*then*') or choice operator ('*or*') to connect messages or changes of role. Messages are either outgoing to another agent in a given role ('⇒') or incoming from another agent in a given role ('⇐'). Figure 2.4 provides a diagrammatical view of these operators. The most basic behaviours are to send or receive messages, and more complex ones can be constructed using combinations of the sequence and choice operators. A set of such behavioural clauses specifies the message passing behaviour expected of a social norm and, in LCC, this is referred to as the interaction framework.

Message input/output or change of role can be governed by a constraint defined using the normal logical operators for conjunction, disjunction and negation. Notice that there is no commitment made in LCC with regards to the choice of constraint language as it depends on the constraint solvers used. However, in the current LCC implementation, constraints are specified as first order predicate calculus. The two options provided by LCC on how agents can satisfy these constraints are as follows [Robertson, '04c]:

- Internally according to whatever knowledge and reasoning strategies it possesses. This is the normal assumption on most MAS, yet it might not always be ideal. Sometimes, it might be preferred not to have the knowledge specifically used for a social interaction internalised within the agents as commonly required (e.g. in cases where knowledge might be inconsistent with the agents' own beliefs). In such cases, LCC offers a second option:

- Externally using a set of Horn clauses defining common knowledge assumed for the purpose of the interaction. This common knowledge can be set as *public* (accessible to all agents participate in the interaction) or *private* (accessible to individual or limited set of agents involved in the interaction). Like the LCC

protocols themselves, the common knowledge is passed between agents along with messages during interaction. Therefore, it is temporary – lasting only as long as the interaction. Further description with regards to this option is provided in the next sub-section.

```
Framework  := {Clause,…}
   Clause  := Role::Def
     Role  := a(Type,Id)
      Def  := Role | Message | Def then Def | Def or Def |
              null ← C
  Message  := M ⇒ Role | M ⇒ Role ← C | M ⇐ Role |
              C ← M ⇐ Role
        C  := Term | ¬C | C ∧ C | C ∨ C
     Type  := Term
        M  := Term
```

Where *null* denotes an event, which does not involve message passing; *Term* is a structured term in Prolog's syntax and *Id* is either a variable or a unique identifier for the agent. The operators ¬, ←, ∧ or ∨ are the normal logical connectives for negation, implication, conjunction or disjunction. `M ⇒ A` denotes that a message, `M`, is sent out to agent `A`. `M ⇐ A` denotes that a message, `M`, from agent `A` is received. The implication operator dominates the message operators, so for example ,
`M ⇒ Agent ← C` is understood as `(M ⇒ Agent) ← C`

Figure 2.3: Syntax of LCC protocol language

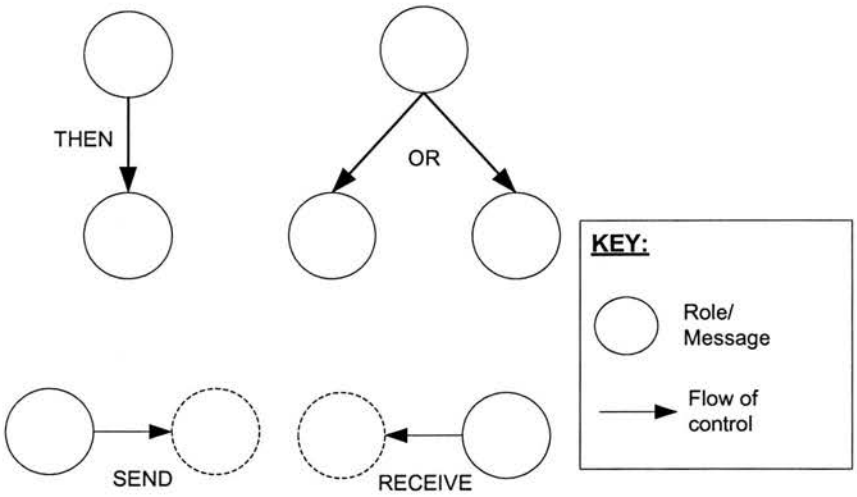


Figure 2.4: Diagrammatical view of LCC operators

Coherence of interaction between agents can be ensured by imposing constraints relating to the messages they send and receive in their chosen roles (see use of C in figure 2.3). Constraints are imposed through the implication operator (marked by '←'), which indicate the requirements or consequences for an agent on the performatives or roles available to it. The clauses of the protocol are arranged so that, although the constraints on each role are independent of others, the ensemble of clauses operates to give the desired overall behaviour [Robertson, '04c]. For example, the LCC protocol of figure 2.5 places two constraints on the variable X: the first (*p(X)*) is a condition on the agent A_1 in role *r1* sending the message *offer(X)* and second (*q(X)*) is a condition on the agent A_2 in role *r2* sending message *accept(X)* in reply. By (separately) satisfying *p(X)* and *q(X)* the agents A_1 and A_2 mutually constrain the variable X.

$a(r1,A_1)::offer(X) \Rightarrow a(r2,A_2) \leftarrow p(X)$ then $accept(X) \Leftarrow a(r2,A_2)$

$a(r2,A_2)::offer(X) \Leftarrow a(r1,A_1)$ then $accept(X) \Rightarrow a(r1,A_1) \leftarrow q(X)$

Figure 2.5: Example of LCC protocol

Although LCC looks different from EI-based framework like ISLANDER, it provides all the representational features of one, as described in detail in [Robertson, '04a]. Other aspects of LCC are further discussed in [Walton and Robertson, '02; Robertson, '03; Robertson, '04c; Robertson, '04b], which are summarised in the following sections of 2.2.2.2 and 2.2.2.3. A discussion on a variant of LCC called Multi-agent Protocol which is implemented in the Java platform is provided in [Walton, '04b; Grando and Walton, '06]. A number of other results based on LCC or similar approaches are described in [McGinnis et al., '03; McGinnis and Robertson, '04; Walton, '04a; Walton and Barker, '04; Lambert and Robertson, '05; McGinnis and Robertson, '05; Grando and Walton, '06; Osman et al., '06].

2.2.2.2 Coordination Mechanism

In LCC, one of the main concerns is for the mechanism used to provide coordination for distributed agent interactions to have as low an impact as possible on the engineering of agents. This can be achieved through a modular mechanism, acting as an intermediary between the agent and the medium used to transmit messages, as depicted in figure 2.6. On the principle, the functionality of the mechanism is similar to the function of governor in EI. However, in LCC, the coordination is managed by the agents themselves who have full control and access to the mechanism.

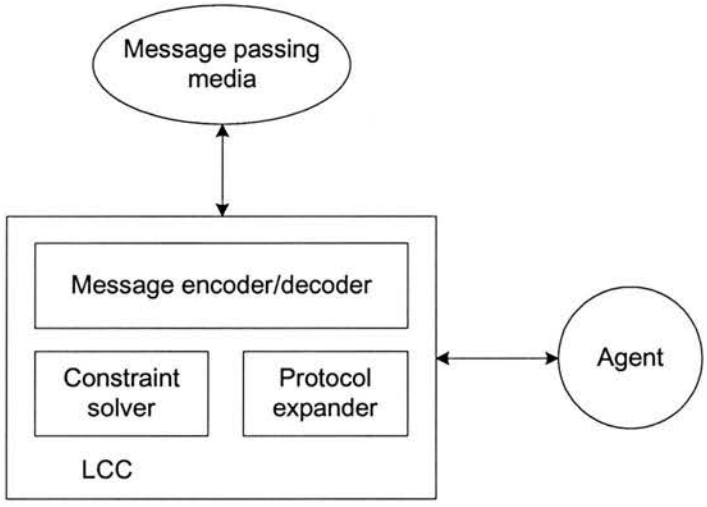


Figure 2.6: Basic architecture of agent interactions in LCC

The module has the following elements:

- A message encoder/decoder for receiving and transmitting messages via whatever message passing media being used to transport messages between agents. For example, if the blackboard-based platform like the Linda tuple space [Carrieno and Gelernter, '89] is being used for inter-agent communication, then the encoder/decoder must be able to read Linda messages and extract the LCC protocol expressions contained within; similarly for other platforms.
- A protocol expander that decides how to expand a protocol received with a message. Detailed specification on this part is provided in sub-section 2.2.2.3.

- A constraint solver capable of deciding whether constraints passed to it by the protocol expander are satisfied.

The existing Prolog-based mechanism for deploying LCC protocols relies on passing the protocol and associated information about the state of the collaboration with messages sent between agents [Robertson, '04c]. This means that the interacting agents do not retain any protocol clause (or clauses if it has multiple roles) appropriate to it. This has the advantage since agents are not required to provide any clause storage, but it works only for interactions that are linear, in a sense that at any given time, only one agent alters the state of the interaction regardless of how many agents are involved in the interactions. An example of a linear interaction is a dialogue between two agents where each agent takes alternate turn in the interaction. An example of a non-linear interaction is an auction involving a broadcast call for bids.

This method of coordination is described in figure 2.7. For ease of discussion, the diagrams depict an interchange between only two agents (Agent 1 and Agent 2), with a message (Message 1) being sent from Agent 1 to Agent 2 and another message (Message 2) being returned in response. The clauses determining the behaviours of the interacting agents are distributed among the agents as the protocol is passed between them. These distributed clauses, which are depicted as clause stores in figure 2.7, describe the state of agents' interactions. Upon receipt of a message, the agents look for their clauses in the clause store. The agents make the necessary update on the respective clauses once they have completed their parts of the protocol. The state of the whole interaction is preserved by the message as it passes between agents.

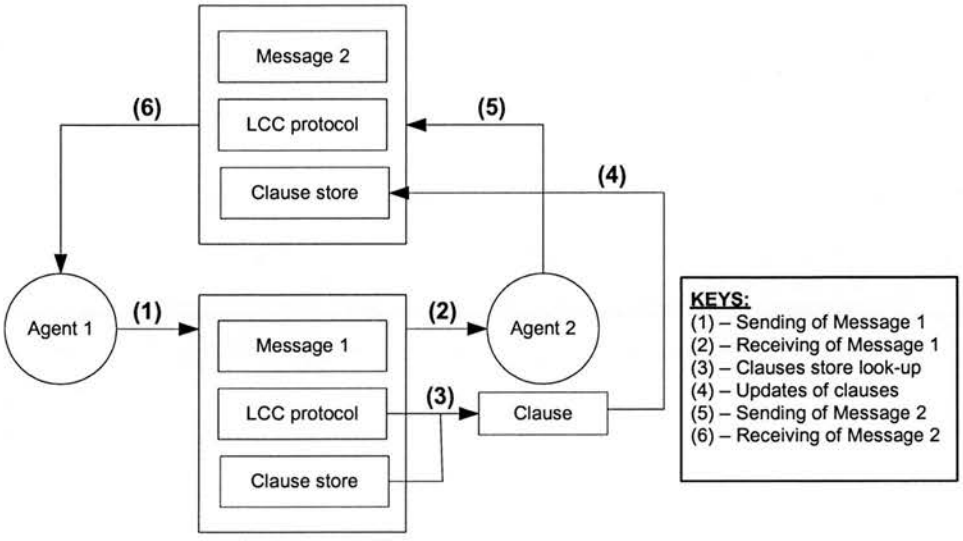


Figure 2.7: Message-passing in LCC

In order to support this method of coordination, the format of messages exchanged among the agents within the LCC is as follows:

i. A message must contain (at least) the following information, which can be encoded and decoded by the sending and receiving mechanisms attached to each agent:

- An identifier, I, for the social interaction to which the message belongs. This identifier must be unique and is chosen by the agent initiating the social interaction.
- A unique identifier, A, for the agent intended to receive the message.
- The role, R, assumed of the agent with identifier A with respect to the message.
- The message content, M, expressed in the dialogical framework shared and understood by the interacting agents.
- The protocol, P, of the form $P := (T,C,K)$ for continuing the social interaction.

    a. T is the interaction state. This is a record of interactions accomplished so far, which indicates the current interaction state for each agent. This is achieved by marking the agent interaction clauses as closed or failed depending on whether they have been communicated successfully. This computational process is performed by the agents themselves after

successfully executing their parts in the protocol. Clauses that have been communicated are enclosed by a '*c*', *c(M)* as illustrated in table 2.1. A protocol term is considered closed, meaning that it has been covered by the preceding interaction, as follows:

closed(c(X)).
closed(A or B) ← closed(A) ∨ closed(B).
closed(A then B) ← closed(A) ∧ closed(B).
closed(X::D) ← closed(D).

b. The second part is a set, *C*, of LCC clauses defining the interaction framework (based on the syntax in figure 2.3).

c. The final part, a set *K*, of axioms consisting of common knowledge as described earlier.

ii. The agent must have a mechanism for satisfying any constraints associated with its clause in the interaction framework. Where these can be satisfied from common knowledge (the set of *K* above), it is possible to supply standard constraint solvers with the protocol. Otherwise, it is the responsibility of the agent.

2.2.2.3 Expansion Engine

Within the LCC, agents themselves are expected to communicate the conventions of the interaction protocol. This is accomplished by the participating agents satisfying the following two engineering requirements.

First, agents are required to share a dialogical framework. This is an unavoidable necessity in any meaningful agent communication. As such, the individual messages and constraints are required to be expressed in an ontology understood by the agents. For the constraints, though their specifications need to be understood by all of the agents involved in the interactions, how they are satisfied is left to the internal reasoning of each individual agent.

Second, agents are required to provide a means to process the received message and its protocol. Given the descriptions about the message format, the basic operation an agent must perform when interacting via LCC is to decide what the next steps for its role

in the interaction should be, using the information carried with the message it receives from some other agent. An agent is capable of conforming to a LCC protocol if it is supplied with a way of unpacking any protocol it receives, finding the next moves that it is permitted to take, and updating the state of the protocol to describe the new state of the interaction. In the current practice, these are achieved by applying rewrite rules of table 2.1 to expand the protocol terms.

The nine rules specified in table 2.1 define the expansion of a single interaction clause. Full expansion of a clause is achieved through exhaustive application of these rules. Rewrite rule 1 expands a protocol clause with head A and body B by expanding B to give a new body, E. The other eight rewrite rules are concerned with the operators in the clause body. A choice operator is expanded by expanding either side, provided the other is not already closed (rewrite rules 2 and 3). A sequence operator is expanded by expanding the first term of the sequence or, if that is closed, expanding the next term (rewrite rules 4 and 5). A message matching an element of the current set of received messages, M_i, expands to a closed message (i.e. marked as *c*(*message*)) if the constraint, *S*, attached to that message is satisfied (rewrite rule 6). A message sent out expands similarly (rewrite rule 7). A null event can be closed if the constraint associated with it can be satisfied (rewrite rule 8). An agent role can be expanded by finding a clause in the protocol with a head matching that role and body B – the role being expanded with that body (rewrite rule 9).

1) $A :: B \xrightarrow{M_i, M_o, P, O} A :: E$	if $B \xrightarrow{M_i, M_o, P, O} E$
2) A_1 or $A_2 \xrightarrow{M_i, M_o, P, O} E$	if $\neg closed(A_2) \wedge A_1 \xrightarrow{M_i, M_o, P, O} E$
3) A_1 or $A_2 \xrightarrow{M_i, M_o, P, O} E$	if $\neg closed(A_1) \wedge A_2 \xrightarrow{M_i, M_o, P, O} E$
4) A_1 then $A_2 \xrightarrow{M_i, M_o, P, O} E$ then A_2	if $A_1 \xrightarrow{M_i, M_o, P, O} E$
5) A_1 then $A_2 \xrightarrow{M_i, M_o, P, O} A_1$ then E	if $closed(A_1) \wedge A_2 \xrightarrow{M_i, M_o, P, O} E$
6) $S \leftarrow M \Leftarrow A \xrightarrow{M_i, M_i - \{M \Leftarrow A\}, P, \varnothing} c(M \Leftarrow A)$	if $(M \Leftarrow A) \in M_i \wedge satisfy(S)$
7) $M \Rightarrow A \leftarrow S \xrightarrow{M_i, M_o, P, \{M \Rightarrow A\}} c(M \Rightarrow A)$	if satisfied(S)
8) $null \leftarrow S \xrightarrow{M_i, M_o, P, \varnothing} c(null)$	if satisfied(S)
9) $a(R, I) \leftarrow S \xrightarrow{M_i, M_o, P, \varnothing} a(R, I) :: B$	if $clause(P, a(R, I) :: B) \wedge satisfied(S)$

A protocol term is said to be closed, meaning that it has been covered by the preceding interaction if the following holds.

$closed(\, c(X)\,)$
$closed(A \text{ or } B) \leftarrow closed(A) \vee closed(B)$
$closed(A \text{ then } B) \leftarrow closed(A) \wedge closed(B)$
$closed(X :: D) \leftarrow closed(D)$

satisfied(S) is true if *S* can be solved using the agent's current knowledge.
satisfy(S) is true if the satisfaction of *S* is derivable from the agent's knowledge.
clause(P,X) is true if clause *X* appears in the interaction protocol *P*.

Table 2.1: Rewrite rules for expansion of a protocol clause

The following describe how the expansion of protocol terms are achieved in LCC:

- An agent with unique identifier, *A*, retrieves a message of the form *(I,M,R,A,P)* where: *I* is a unique identifier for the interaction; *M* is the message; *R* is the role assumed of the agent when receiving the message; *A* the agent's unique identifier; and *P* the attached protocol consisting of *T*, the dialogue state; a set of dialogue clauses, *C*; and a set of axioms, *K*, defining knowledge pertaining to the subject matter of the interaction. The message is added to the set of messages currently under consideration by the agent – yielding the message set $M_i \in M$.

- The agent extracts from P the interaction clause, $C_i \in C$, determining its part of the interaction.
- The rewrite rules of table 2.1 are applied to give an expansion of C_i in terms of protocol P in response to the set of received messages, M_i, producing: a new interaction clause C_n, an output message set O_n and remaining unprocessed messages M_n (a subset of M_i). These are produced by applying the protocol rewrite rules in table 2.1 exhaustively to produce the sequence:

$$\left(C_i \xrightarrow{M_i, M_{i+1}, P, O_i} C_{i+1}, C_{i+1} \xrightarrow{M_{i+1}, M_{i+2}, P, O_{i+1}} C_{i+2}, \ldots, C_{n-1} \xrightarrow{M_{n-1}, M_n, P, O_{n-1}} C_n\right)$$

- The agent's original clause, C_i, is then replaced in P by C_n to produce the new protocol, P_n.
- The agent can then send the messages in set O_n, each accompanied by a copy of the new protocol P_n.

2.3 Distributed Problem Solving Environment

In a distributed problem solving environment, sub-problems are interdependent and overlapping [Decker et al., '88], so agents working in the environment must carefully coordinate their local problem solving actions which can only be achieved through proper agent interactions. These interactions allow interdependence of the sub-problems due to the relationships that exist between them to be adequately resolved. These relationships can be associated to two basic situations related to the natural decomposition of domain problem solving into sub-problems to be solved individually by the agents [Lesser, '99]. The descriptions of the situations are as follows:

Similar or Overlapping Sub-problems Situation. In this situation, different agents have either alternative methods or data that can be used to generate a solution given a set of similar or overlapping sub-problems. For example, in a distributed situation assessment application, overlapping sub-problems occur when different agents are interpreting data

from different sensors (independent information sources) that have overlapping sensor regions (cover similar information).

Sub-problems are Part of a Larger Problem Situation. In this situation, a form of interdependence occurs when a number of sub-problems are part of a larger problem in which a solution to the larger problem requires that certain constraints exist among the solutions to its sub-problems. For example, in a distributed expert system application involving the design of an artefact where each agent is responsible for the design of a different component (sub-problem), there are constraints among these sub-problems that must be adhered to if the individual component designs will mesh together into an acceptable overall design. This situation also includes the case where the results of one sub-problem are needed to solve another.

Besides these two, there exists another situation where the interdependencies among sub-problems are not inherent to the problem domain. This occurs when it is not possible to decompose the problem into a set of sub-problems to allow a perfect fit between the computational requirements for effectively solving each sub-problem and the agents to solve them. An example of this type of constraint is insufficient local information or resources for an agent to completely or accurately solve the assigned sub-problems through its own processing. This might lead to the creation of shared agent plans so that the use of scarce resources can satisfy multiple objectives of the agents or the reconfiguration of resources to better meet the competing needs of agents.

The sub-problems handled by the distinct agents can be modelled using a Distributed Constraint Satisfaction Problem formalism. This formalism, an extension of Constraint Satisfaction Problem framework, is developed to accommodate the needs of distributed problem solving environments. The definitions for CSP and DCSP are given as follows:

Definition 2.1: Constraint Satisfaction Problem (CSP)

A CSP is a problem composed of [Tsang, '93]:

- a finite set of variables, $V=\{V_1,\ldots,V_k\}$, and each variable $V_i \in V$ is associated with
  - a finite domain of values, $D=\{D_1 \ldots D_k\}$, and
  - a set of constraints, $C=\{C_1,\ldots,C_m\}$ which restricts the values that the variables can simultaneously take

Definition 2.2: Distributed Constraint Satisfaction Problem (DCSP)

DCSP is defined abstractly as consisting of the following three components [Yokoo et al., '98]:

- an agent set $A = \{A_1, A_2,\ldots, A_n\}$, finite, non-empty set
- each agent $A_i \in A$ has a finite set of k variables $V_1, V_2,\ldots, V_k$, and each variable is associated with a finite domain of values $D_1, D_2,\ldots, D_k$, that can be assigned to the variables
- there exist two kinds of constraints over the variables among the agents that defines the permissible subsets of assignments to the variables:
  - Intra-agent constraints, between variables of the same agent
  - Inter-agent constraints, between variables of different agents

This formalism is further refined in the Multi-agent Agreement Problem (MAP), which is a special class of DCSP. The major difference between the two is that the former allows a variable to be shared among a set of agents (participants) while the latter assigns each variable to a unique agent. As it is specifically intended to model "agreement", the MAP requires the constraints between variables belonging to different agents to be limited to equality constraints. DCSP, on the other hand admits general inter-agent constraints.

The motivation for introducing the MAP representation with shared variables is to conveniently and explicitly capture problems where multiple agents are involved in a joint decision. This is a feature of many distributed problem solving domains where each agent brings its own private constraints to bear on the decision, yet agents must come to an agreement.

Definition 2.3: Multi-Agent Agreement Problem (MAP)

MAP can be defined as follows [Modi and Veloso, '04; Davin and Modi, '06]:

- $A = \{A_1, A_2, \ldots, A_n\}$ is a set of agents
- $V = \{V_1, V_2, \ldots, V_m\}$ is a set of variables
- $D = \{D_1, D_2, \ldots, D_k\}$ is a set of values. Each value can be assigned to any variable
- $participants(V_i) \subseteq A$ is a set of agents assigned the variable V_i. A variable assigned to an agent means it has (possibly shared) responsibility for choosing its value
- $vars(A_i) \subseteq V$ is the set of variables assigned to agent A_i
- For each agent A_i, C_i is an intra-agent constraint that evaluates to true or false. It must be defined only over variables in $vars(A_i)$
- For each variable V_i, an inter-agent "agreement" constraint is satisfied if and only if the same value from D is assigned to V_i by all the agents in $participants(V_i)$

An assignment of values to variables is valid (sound) iff it satisfies both inter-agent and intra-agent constraints. An assignment is complete iff every variable in V is assigned some value. The goal is to find a valid and complete assignment. For example, figure 2.8 provides an abstract illustration concerning the interdependency that involves four variables (i.e. V_1, V_2, V_3 and V_4) and three agents (i.e. agent A, agent B and agent C), in which each node represents a variable, and each arc represents a local constraint between variables represented by the end points of the arc. The intra-agent constraints of each agent are varied in terms of constraint density, in which agent B has a highly constrained problem while agent C has a least constrained one. Since agents are distributed in different locations or in different processes, each agent only knows the partial problem associated with those constraints in which it has variables. A global solution then consists of a complete set of the overlapping partial solution of each agent. Interaction among agents is necessary and important for solving this problem, since each agent only knows its variables, variable domains and related inter-agent and intra-agent constraints. A

solution S, is an instantiation for all variables that satisfies all intra-agent and inter-agent constraints.

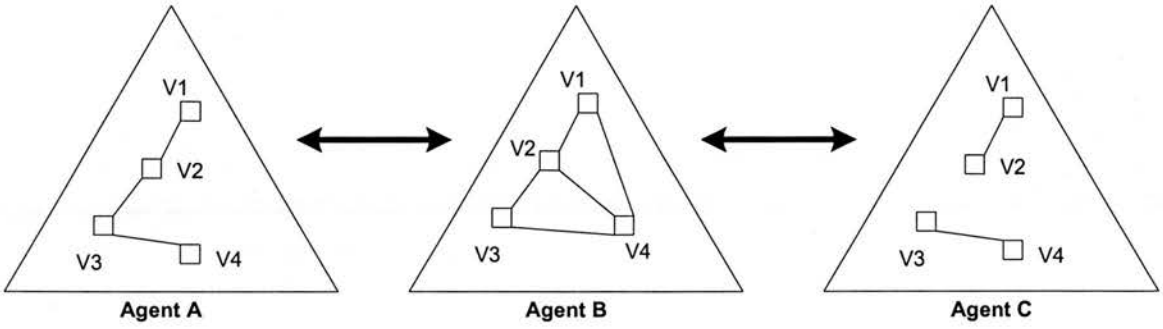


Figure 2.8: Variable interdependency in distributed problem solving

2.4 Over-Constrained Problems

A CSP consists of a finite number of variables, each having a finite and discrete set of possible values, and a set of constraints over these variables. A solution to a CSP is an instantiation of all variables for which all the constraints are satisfied. Though powerful, the CSP schema presents some limitations. In particular, all constraints are considered mandatory and need to be fully satisfied. However, in many real-world problems, it is often the case that there exists no consistent instantiation of variables that satisfies all constraints. This leads to unsolved problems. These problems are said to be over-constrained: any complete assignment of variables violates some defined constraint of the CSP [Meseguer et al., '03; Zhou et al., '05]. An over-constrained problem is illustrated in figure 2.9: the Robot Clothing Problem [Freuder and Wallace, '92]. The nodes in the graph represent the three variables – *shoes*, *shirt* and *slacks* – representing the items of clothing that must be chosen. Each node is also labelled with a set of values for the corresponding variables, i.e. the domain of each variable. The arcs that connect the variables are labelled with the legal combinations of values for each of the variables, i.e. the constraints between the variables. Since the conventional formulation of CSPs requires *all constraints* to be satisfied, visibly, this problem is over-constrained as it admits no solution.

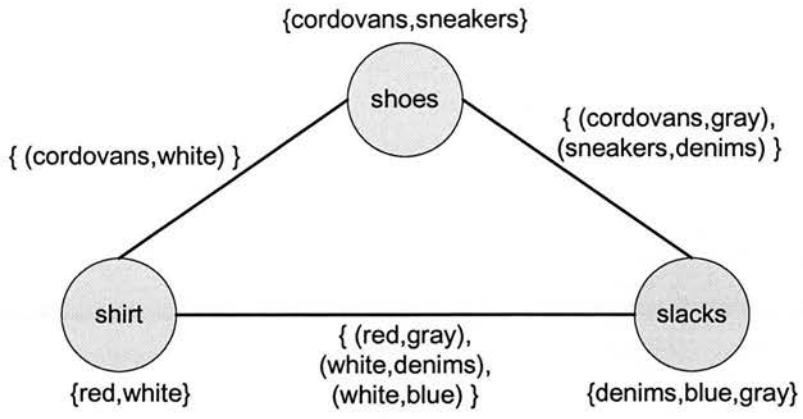


Figure 2.9: Example of an over-constrained problem

In practice however, it is sometimes the case that certain constraints can be violated occasionally, or weakened to some degree. As conventional CSP techniques lack the mechanisms to accommodate such a notion of constraint handling, this gives rise to the establishment of a niche research area within the constraint satisfaction research field focusing on approaches to solve over-constrained problems. These approaches include:

Extended CSP. Constraints in the conventional CSP scheme are *crisp*, in which they can only be either in two possible states – completely satisfied or completely violated. In order to address this rigidity, several models have been devised to extend the existing CSP scheme to enable it to accommodate different constraint representations that include non-crisp constraint forms like fuzziness, probabilities and weights [Meseguer et al., '03].

- In the fuzzy model, constraints are represented by fuzzy relations [Dubois et al., '96]. In this model, constraint satisfaction becomes a matter of degree. The degree in which this relation is satisfied is given by a membership function from the interval [0, 1], where 1 means complete satisfaction and 0 complete violation. The satisfaction degree for each possible value assignment is computed, and a solution is the value assignment with maximum satisfaction degree.

- In the weighted model, each constraint is labelled with a weight, which represents the cost (or penalty) that exists if the constraint is violated. The cost of a complete assignment is the addition of costs of all constraints instantiated by that assignment. A solution is the value assignment with minimum cost.

- In the probabilistic model, each constraint is labelled with a probability of presence, assumed independent of the presence of other constraints. A solution is the value assignment with maximum probability of being a solution to the real problem.

Partial Constraint Satisfaction Problem. In the partial Constraint Satisfaction Problem (partial CSP) model, constraints are represented by crisp relations. The scheme proposed in [Freuder and Wallace, '92] is an interesting extension to CSP, which allows the relaxation and optimisation of over-constrained problems via the weakening of the original CSP. In this scheme, a general model of partial constraint satisfaction is proposed that provides comparison with alternative problems rather than alternative solutions. It is suggested that partial satisfaction of a problem, *P*, should be viewed as a search through a space of alternative problems for a solvable problem "close enough" to *P*. Freuder and Wallace argue that a full theory of partial satisfaction should consider not merely how a partial solution requires us to violate or vitiate constraints, but how the entire solution set of the problem with these altered constraints differs from the solution set of the problem with which we started. This scheme provides the basis of the proposed approach to address distributed over-constrained problems that lead to the brittleness of the interaction protocol. Further details with regard to this scheme are provided in the next section and the notion of a brittle agent interaction is described using an example in chapter 3.

Constraint Hierarchies. In this model constraints are divided into a hierarchy of levels, according to their relative importance. This model, which only considers crisp constraints, provides a framework to define a constraint hierarchy as a finite collection of constraints labelled with a level of strength or preference, i.e. hard and soft constraints [Borning et al., '92]. While the hard (required) constraints must hold, the soft (preferential) constraints should be satisfied as much as possible depending on the criteria used. A solution to an over-constrained problem then is an assignment of values to variables that best satisfies the constraints and respecting the associated hierarchy.

Though there exist a number of well-established approaches for solving over-constrained problems, partial CSP is chosen as it fits well for the interaction protocol environment – no further assumptions are needed with regards to the formalism and criteria used by the heterogeneous and distributed agents concerning the constraints communicated between them. In partial CSPs, constraints are represented as crisp relations, which have been generally accepted as a natural formalism to specify many kinds of real-life problems. As such, agents face an over-constrained situation and fail to expand their parts in the protocol led interaction are neither obligated nor required to revise the formalisation of their local problems. In addition, partial CSP has also been extended to support the solving of distributed, over-constrained problem. This new, extended scheme is known as the distributed partial CSP [Hirayama and Yokoo, '97; Yokoo, '01]. In the other approaches used to address over-constrained problems (i.e. extended CSP or Constraint Hierarchies), there is a need to provide an additional formalism to appropriately represent the extended mechanism used in handling the constraints (i.e. fuzzy, probability or hierarchy). Integrating these approaches with LCC will require a major revision on the current distributed interaction protocol system of LCC to accommodate this need. This, however, is a separate research issue that is beyond the scope of our current research. We are interested in a mechanism to coordinate and compute the weakening of the original CSPs among the interacting agents faced with an over-constrained problem which causes an interaction to break.

2.5 Partial Constraint Satisfaction Problems

A partial CSP requires the weakening of a problem in order to accept more solutions. Essentially, in partial CSP, the focus is on relaxing the original CSP so that a satisfactory solution can be found [Freuder, '90]. For a given CSP, one might relax it based on the following four options [Freuder and Wallace, '92], and the example of the over-constrained problem in figure 2.9 is used to illustrate each option:

1. Enlarging a variable domain (e.g. buying a new shirt)
2. Enlarging a constraint domain (e.g. deciding that certain shoes do, after all, go with a certain shirt).
3. Removing a variable (e.g. deciding not to wear shoes at all).
4. Removing a constraint (e.g. ignore clashes between shoes and shirts).

However, all of these options can be considered in terms of the basic process of enlarging constraint domains (i.e. option 2). For instance, option 1 of enlarging a variable domain is the same as enlarging the domain of a constraint since a variable domain can be defined as a unary constraint. Removing all the constraints on a variable is equivalent to removing the variable of option 3, while enlarging a binary constraint until it contains all pairs of values in the specified domains for the two variables is tantamount to removing the constraint as defined in option 4.

Formally, a partial CSP can be viewed as a partially ordered set of CSPs, with a common root. The root is the original CSP. The rest of the nodes in the graph are CSPs obtained from the original one through a sequence of relaxation operations, as illustrated in figure 2.10 [Yang and Fong, '92]:

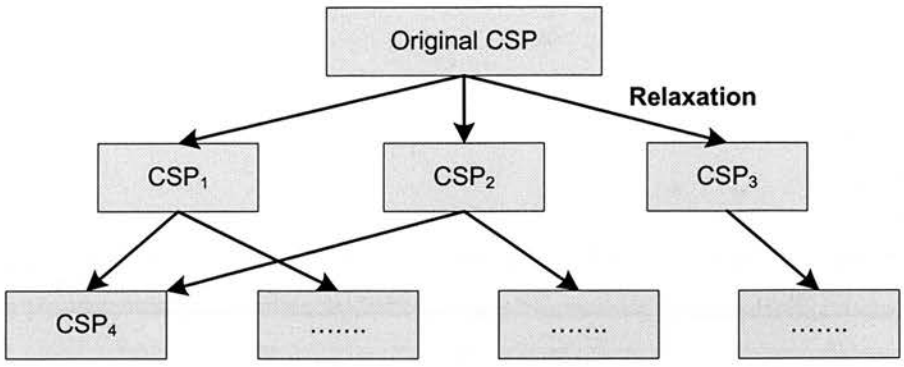


Figure 2.10: The problem space of partial constraint satisfaction

Given two CSPs in the graph, one can measure the distance between them, by associating a partial CSP with a metric. The metric might measure the difference in the number of solutions, the number of added domain values, or it might measure the number of missing (or relaxed) constraints. Solving a partial CSP then becomes a problem of finding a solution of a relaxed CSP within the space of partial CSP, so that the distance metric between the solution of the relaxed CSP and the ideal solution of the original CSP is within some acceptable bound. Two special bounds have been established to ensure the space of partial CSPs is restrained. The first is a sufficient bound, which specifies that a solution to a relaxed CSP is good enough if the metric distance between the solution and the ideal solution is within this bound. The second one is the necessary bound which specifies that the space of CSPs under consideration must all contain solutions that are within the bound.

Definition 2.4: partial Constraint Satisfaction Problem

A partial CSP can be formally described as a triple [Freuder and Wallace, '92]:

$\langle (P, U), (PS, \leq), (M, (Necs, Suff)) \rangle$, where

- P is an original CSP, U is a set of 'universes', i.e., a set of potential values for each variable in P
- $(PS, \leq)$ is a problem space, where PS is a set of CSPs (including P), and $\leq$ is a partial order over PS

- M is a distance function over the problem space, and (Necs,Suff) are necessary and sufficient bounds on the distance between P and some solvable member of PS

A *solution* to a partial CSP is a soluble problem P' from the problem space and its solution, where the distance between P and P' is less than Necs. Any solution will suffice if the distance between P and P' is not more than Suff, and all search can terminate when such a solution is found. An *optimal* solution to a partial CSP is a solution in which the distance between P and P' is minimal, and this minimal distance is called the optimal distance.

The partial-order defined over the problem space PS, is defined in terms of the set of solutions to problems. Specifically, $P_1 \leq P_2$ iff $sols(P_1) \supseteq sols(P_2)$, where $sols(P_1)$ and $sols(P_2)$ denotes the set of solutions to problem P_1 and P_2 respectively. $P_1 \leq P_2$ can be read as "P_1 is obtained by weakening the constraints in P_2". As the problem is weakened, the constraints in the problem allow more consistent assignments and, as a consequence, the set of solutions may increase.

The manner in which a weakened problem is evaluated depends on the distance metric, M, that is used. A number of metrics have been proposed [Bistarelli et al., '04], and these include solution subset distance, augmentation distance and Max-CSP distance, which are described as follows:

- Solution subset distance – The distance metric is defined as the number of solutions not shared between the problems P and P'. When $P' \leq P$, this metric reflects the number of solutions that have been introduced due to the relaxation of the original problem P.

- Augmentation distance – The distance metric is slightly different to solution subset distance. It counts the number of constraint values that are not shared by problems P and P'. This represents the number of augmentations to the constraints in problem P that are required to reach its relaxation P'.

- Max-CSP distance – This is the most well-studied distance metric of the three. It involves finding a solution that violates the minimum number of constraints

in the problem. The metric is normally defined as the number of constraints that are violated.

Considering that the partial CSP approach is applied to resolve the over-constrained CSP of figure 2.9, and given that a simple distance function is adopted (i.e. solutions involving the smallest number of augmentation is preferred), then figure 2.11 provides five equally good weakened problems obtained. For each weakened problem, only one of the constraints is chosen to receive one extra pair of values as illustrated in figure 2.11. In the figure, the notation $C_{x,y}$ is used to indicate the constraints between variables x and y, that is the legal combination of values for each of the variable.

The partial CSP scheme has been extended by Hirayama and Yokoo for distributed environments and is known as distributed partial CSP.

Definition 2.5: distributed partial Constraint Satisfaction Problem

A distributed partial CSP consists of [Hirayama and Yokoo, '97; Yokoo, '01]:

- A set of agents (problem solvers), 1, 2,..., m
- $\langle (P_i, U_i), (PS_i, \leq), M_i \rangle$ for each agent i
- (G, (Necs, Suff)), where

For each agent i, P_i is an original CSP (a part of an original distributed CSP), and U_i is a set of *universes*, i.e. a set of potential values for each variable in P_i. Furthermore, $(PS_i, \leq)$ is called a *problem space*, where PS_i is a set of (relaxed) CSPs including P_i, and $\leq$ is a partial order over PS_i. Also, M_i is a locally-defined *distance function* over the problem space. G is a *global distance function* over distributed problem spaces, and (Necs, Suff) are necessary and sufficient bounds on the global distance between an original distributed CSP (a set of P_is of all agents) and some solvable distributed CSP (a set of solvable CSPs of all agents, each of which comes from PS_i).

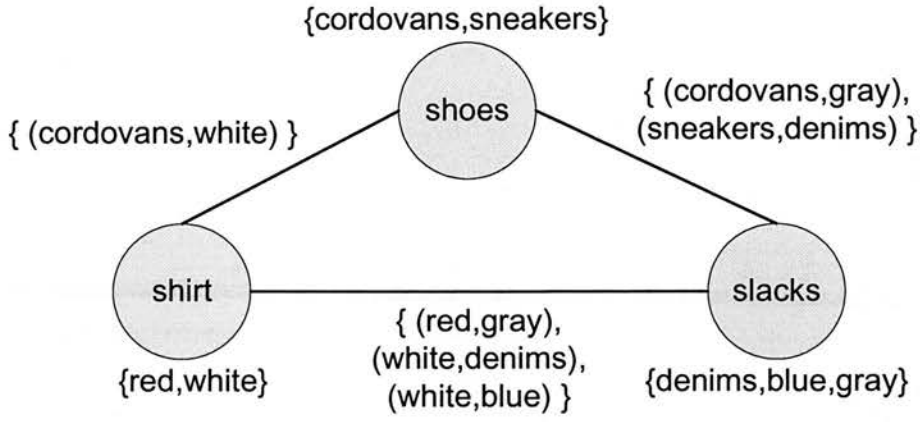
A solution to a distributed partial CSP is a solvable distributed CSP and its solution, where the global distance between an original distributed CSP and the solvable distributed CSP is less than Necs. Any solution to a distributed partial CSP will suffice if

the global distance between an original distributed CSP and the solvable distributed CSP is not more than Suff, and all search can terminate when such a solution is found.

Given a distributed partial CSP scheme, we are interested in studying on how this established theoretical model can be interpreted using the LCC in order to address the over-constrainedness problem which causes the brittleness of agent interaction as described earlier.

2.6 Chapter Summary

In this chapter, we provided a detailed review on two areas deemed important to our research work; agent interaction and distributed, over-constrained CSP. For the former, we focused on two objective-based approaches to agent interaction namely EI and LCC, described respectively in sections 2.2.1 and 2.2.2. For the latter, we begun by presenting an overview on the distributed problem solving environment in section 2.3, that include formal definitions of CSP, DCSP and MAP. We then presented a discussion on over-constrained problem in section 2.4, followed by an overview on three approaches for solving the problem; extended CSP, partial CSP and constraint hierarchies. Furthermore, we also discussed the reasons of choosing partial CSP instead of the other approaches in our research. Finally, in section 2.5, we presented a formal definition of partial CSP and distributed partial CSP, including the distance metrics that could be employed by the approach.



Over-constrained problem

Examples of weakened problems (additional pairs are bold and in different font):

1) $C_{shirt,slack}$: {(red,gray),(white,denims),(white,blue)}
   $C_{shoes,slacks}$: {(sneakers,denims),(cordovans,gray)}
   $C_{shirt,shoes}$: {(cordovans,white), **(sneakers,white)**}
   Solution: Shirt=white, shoes=sneakers, slacks=denims

2) $C_{shirt,slack}$: {(red,gray),(white,denims),(white,blue), **(white,gray)**}
   $C_{shoes,slacks}$: {(sneakers,denims),(cordovans,gray)}
   $C_{shirt,shoes}$: {(cordovans,white)}
   Solution: Shirt=white, shoes=cordovans, slacks=gray

3) $C_{shirt,slack}$: {(red,gray),(white,denims),(white,blue)}
   $C_{shoes,slacks}$: {(sneakers,denims),(cordovans,gray), **(cordovans,blue)**}
   $C_{shirt,shoes}$: {(cordovans,white)}
   Solution: Shirt=white, shoes=cordovans, slacks=blue

4) $C_{shirt,slack}$: {(red,gray),(white,denims),(white,blue)}
   $C_{shoes,slacks}$:{(sneakers,denims),(cordovans,gray), **(cordovans,denims)**}
   $C_{shirt,shoes}$: {(cordovans,white)}
   Solution: Shirt=white, shoes=cordovans, slacks=denims

5) $C_{shirt,slack}$: {(red,gray),(white,denims),(white,blue)}
   $C_{shoes,slacks}$: {(sneakers,denims),(cordovans,gray)}
   $C_{shirt,shoes}$: {(cordovans,white), **(cordovans,red)**}
   Solution: Shirt=red, shoes=cordovans, slacks=gray

Figure 2.11: Example of partial CSP application to solve an over-constrained problem

Chapter 3

Interaction Formalisation and Over-Constrained Problems

This chapter provides a discussion on the interaction model, formalised and executed using the LCC, for the scenario of section 1.3 described in chapter 1. Using this example, brittleness of the interaction model due to over-constrained problems will be described in detail. This chapter also provides an overview of how constraint relaxation based on the distributed partial CSP approach is able to address the problem.

3.1 Customer/Vendor Scenario

The scenario described in chapter 1 involves a series of interactions between two agents (i.e. customer and vendor) over a number of computer parts. This interaction can be described as bilateral multi-issue negotiations [Fatima et al., '03; Heifetz and Ponsati, '04]. There are two ways a multi-issue interaction can be handled – the agents can communicate all the issues together (i.e. a bundle) or one after the other (i.e. issue-by-issue). Assuming that in this particular situation the agents decided on the latter, then the problem can be formalised as an incremental Multiagent Agreement Problem (MAP) [Modi and Veloso, '05], where the process of reaching a mutual agreement requires each attribute (e.g. configuration options and pricing constraints) of the computer to be communicated on an attribute-by-attribute basis among the interacting agents.

A simple LCC-based interaction protocol for the scenario is described in figure 3.1. There are two types of agent: a vendor agent and a customer agent. No limit is placed on the number of interaction instances (i.e. dialogues) that may occur, although each such dialogue will be constrained by the LCC protocol.

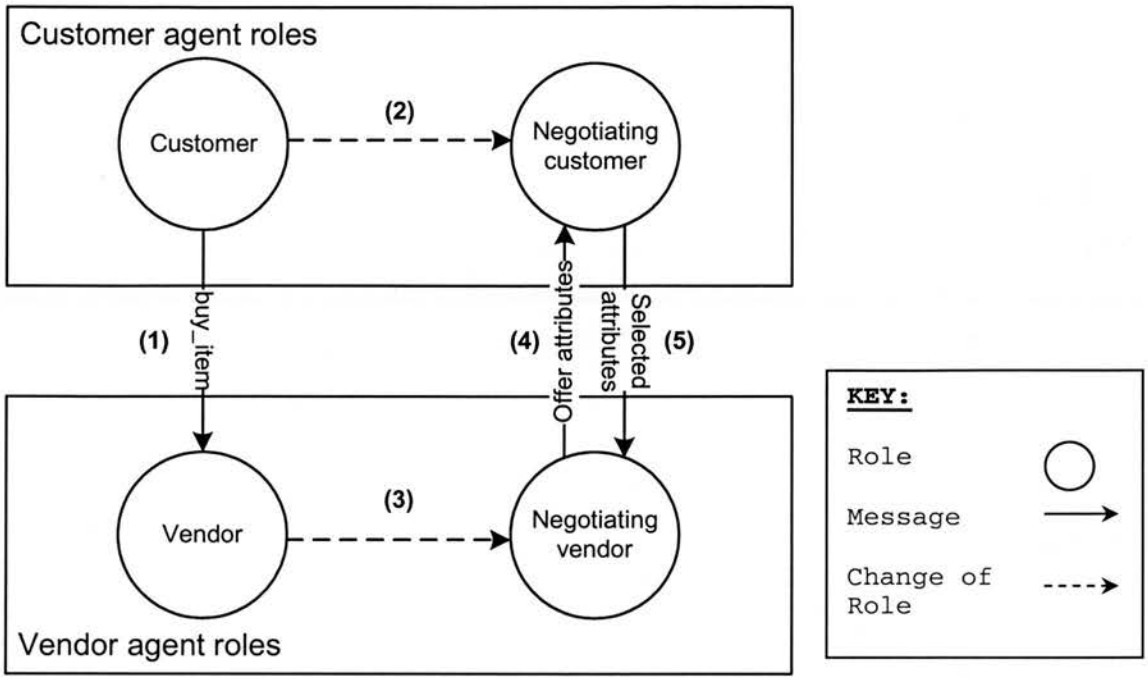


Figure 3.1: Roles and interaction diagram

Assuming that the customer agent has already obtained the necessary interaction information from a service broker, the agent (in the role of customer) may send a request to buy a computer to a selected vendor agent (1), and then assumes the role of negotiating customer (2). On the other hand, upon receipt of a request to buy a computer (1), the agent (in the role of vendor) may assume the role of negotiating vendor (3). In these roles, the agents take turn to make offering (4) and provide selection (5) on each attribute values of the computer to be purchased.

The interaction protocols between the vendor and customer agents are defined by expressions 1-4, in table 3.1. In expression 1, a customer C, can send a request to vendor V, to buy an item X that the customer needs and believes the vendor sells. The customer can then take the role of negotiator with the vendor. Expression 2 consists of clauses to define a negotiating customer with a set S, of negotiated attributes of the desired item X. When assuming this role, the agent receives an offer of a new attribute A, and accepts that (continuing in the negotiating role with A added to S). In expression 3, a vendor V, receives a request from a customer C, to buy an item X; then takes the role of negotiator with the customer over the attribute set S, which applies to that item. In expression 4, a negotiating vendor with a set S, of negotiable attributes of the desired item X takes the

first element A of S and offers it to the customer for acceptance (continuing then in its negotiating role with the remaining attributes, T) until S is empty. A more elaborate interaction protocol for the scenario can be obtained in [Robertson, '04c], which also provides a comprehensive discussion on the operation of the protocol using concrete examples. This part of the thesis, on the other hand, demonstrates how the protocol might fail due to over-constrained problems.

```
a(customer,C)::
   ask(buy(X)) ⇒ a(vendor,V) ← need(X) ∧ sells(X,V) then
   a(neg_customer(X,V,[]),C)                                    (1)

a(neg_customer(X,V,S),C)::
   offer(A) ⇐ a(neg_vendor(X,C,_),V) then
   accept(A) ⇒ a(neg_vendor(X,C,_),V) ← acceptable(A) then
   a(neg_customer(X,V,[att(A)|S]),C)                            (2)

a(vendor,V)::
   ask(buy(X)) ⇐ a(customer,C) then
   a(neg_vendor(X,C,S),V) ← attributes(X,S)                     (3)

a(neg_vendor(X,C,S),V)::
   offer(A) ⇒ a(neg_customer(X,V,_),C)← S=[A|T] ∧ available(A) then
   accept(A) ⇐  a(neg_customer(X,V,_),C) then
   a(neg_vendor(X,C,T),V)                                       (4)
```

Table 3.1: LCC protocol for the given scenario

Realising Inter-Agent Constraints. The protocol ensures coherence of interaction between agents by imposing constraints relating to the message they send and receive in their chosen roles. The clauses of a protocol are arranged so that, although the intra-agent constraints on each role are independent of others, the ensemble of clauses operates to give the desired overall behaviour, which involves setting the inter-agent constraints. For instance, as defined in expressions 2 and 4, the protocol places two constraints on each attribute (A) from the set of attributes (S) of the computer to be purchased: the first constraint (*available(A)*) of expression 4 is a condition on the agent adopting the role of negotiating vendor of sending the message *offer(A)* and the second constraint

(*acceptable(A)*) of expression 2 is a condition on the agent adopting the role of negotiating customer sending the message *accept(A)* in reply. By (separately) satisfying these intra-agent constraints imposed on the interaction protocol terms associated with the attribute, the agents mutually constrain the attribute A, and consequently realise the corresponding inter-agent constraints.

Specifying and Satisfying Intra-Agent Constraints. The agents involved in the protocol must be capable of satisfying the constraints that they impose. Though in LCC no commitment is made with regards to how the agents satisfy the constraints imposed on the assumed roles, for the purpose of this research, the formalisms used in specifying the constraints are expected to be standard for all agents, in which case they are shared among all agents (and propagated with the protocol). In this work, a finite-domain formalism is used to assign a range of valid domain values that can be assigned to the set of variables V. This means, that given a set of variables $V=\{V_1,..,V_n\}$, there exists a set of domain values $D=\{D_1,..,D_n\}$: where each $D_i(1 \leq i \leq n)$ is a set of possible finite-domain values for variable V_i. This means the value for the variable V_i must be in the given finite-domain D_i [Fruhwirth, '98]. More precisely, if D_i is an:

- *Enumeration domain*, $List=\{value_1,...,value_k\}$, then the value for V_i is a ground term that appears in *List*. For instance, given a list of *Colour={Red, Blue, White}*, then the value for $V_i \in Colour$.
- *Interval domain*, *{Min..Max}*, then the value for V_i is a ground term between *Min* and *Max* inclusive. For instance, given *Weight={50..80}*, then the value for $V_i \in Weight$.

These specifications constitute what we call *unary constraints*. *Binary constraints* over pairs of variables could also be represented using finite-domain constraint specification that reflects the dependency relationship between them. For instance, the finite-domain constraint imposed on variable V_i can be specified as an equation in the form of $V_i=\{1000+((V_{i-1}/14)*100)+((V_{i-2}-40)*10)\}$, which constitutes two parts; a fixed constant of 1000, and a non-fixed component that depends on the available finite-domain values of variables V_{i-1} and V_{i-2}.

Any standard constraint logic programming mechanism could be used for this purpose. In our case, since LCC is implemented in SICStus Prolog, the finite-domain constraint solver available in SICStus Prolog (i.e. clp(FD)) [SICS, '99] is used to handle the finite-domain constraints imposed on variables contained in the protocol. The definition of the constraint handling clauses (which typically would be private to each agent) imposed on these variables are adopted from the syntax, and the predicates of SICStus Prolog clp(FD) library. A subset of the clp(FD) predicates, especially those needed for the purpose of specifying the constraints of the given scenario is introduced in the following. The examples on how these are utilised by the customer and vendor agents in composing their individual finite-domain constraint clauses for the computer attributes are given in section 3.2.

- Domains of variables will be set of integers or atoms. The predicate **in** is used to state the domain of a variable, written as **Att in Set,** where **Att** is a variable name and **Set** can be:

{Integer1, Integer2,...} or **{Atom1, Atom2, ...}**	Set of enumerated integers or atoms
Term1..Term2	Set of continuous integers between **Term1** and **Term2,** or the constant **inf** (for lower infinity) or the constant **sup** (for upper infinity)
Set1 \/ Set2	Union of **Set1** and **Set2**
Set1 /\ Set2	Intersection of **Set1** and **Set2**
\ Set	Complement of **Set**

- Finite-domain constraints can also be composed of dependency relationships that exist between the variables maintained by the agents. These relationships can be represented **Att Relation Expr,** where **Att** is a variable name and **Expr** is an arithmetic expression in one of the following forms:
  i. A grounded variable on which **Att** is dependent.
  ii. A constant (numeric or non-numeric) on which **Att** is dependent.
  iii. A set of variables and/or constants connected with the mathematical operators *, /, - , +, mod, div on which **Att** is dependent.

and **Relation** can be:

#=	Equal
#\=	Not equal
#<	Less than
#>	Greater than
#=<	Less or equal
#>=	Greater or equal

We chose to limit our research to the finite-domain constraint problem because of the following reasons [Schulte and Carlson, '06]:

- Practical relevance – the most common constraint solving problem only involves variables that are discrete and have finite domains.

- Existence of known principles and techniques – the research on constraints over finite domains is a main-stream research within the constraint satisfaction field. As known principles and techniques have been conceived and documented for finite domains, they stimulate the development of many tools to support finite-domain constraint computation in practice. For instance, in this research, the clp(FD) library of SICStus Prolog is utilised in the implementation.

Accommodating Distributed Finite-Domain Constraints Solving. The finite-domain constraints on variables, individually defined by the distinct agents on the variables of the MAP to be solved, are entirely separate from each other and private to each agent. So, when an agent locally solves a set of finite-domain constraints pertaining to a variable, it will not propagate to the other agents unless carried by the protocol. In order to accommodate this requirement, LCC is equipped with means of propagating finite-domain constraint solving across agents' interactions. A formal model to describe agent interactions for a distributed finite-domain constraint solving, via a LCC-based protocol, is provided as follows:

$$\sigma(p, G_p) \leftrightarrow \left[\Omega \overset{p}{\ni} \langle S, V \rangle \wedge i(S, \varnothing, V, S_f, V_f) \wedge \left(k_p(S_f) \vdash G_p \right) \right] \tag{5}$$

$$i(S, M_i, V_i, S_f, V_f) \leftrightarrow S = S_f \vee \left(\begin{array}{l} S \overset{s}{\supseteq} S_p \wedge \\ \text{apply\_ranges}\,(V_i, S_p, S'_p) \wedge \\ S'_p \xrightarrow{M_i, S, M_n} S''_p \wedge \\ \text{update\_ranges}\,(S''_p, V_i, V_n) \wedge \\ S''_p \overset{s}{\cup} S = S' \wedge \\ i(S', M_n, V_n, S_f, V_f) \end{array} \right) \tag{6}$$

This model compactly describes all the protocol handling features provided by LCC as described in detail in section 2.2.2, with some additional features to accommodate the handling of distributed finite-domain constraint solving, and the means to propagate this across agents. The model is composed of components which can be described as follows:

- p is a unique identifier for an agent and G_p is a goal agent p wants to achieve.

$$\Omega \overset{p}{\ni} \langle S, V \rangle \leftrightarrow S \in \Omega \wedge S = m(\varnothing, P_g, K) \wedge c(p, P_g, K, V)\text{, where} \tag{7}$$

- Ω, is the set of all initial interaction states available to agents. An interaction is initiated when agent p selects the appropriate initial interaction state, S, pertaining to a particular MAP to be solved. S, is a protocol structure consisting of the interaction

model used to coordinate the agents (i.e. P_g) in cooperatively solving the MAP; the interaction specific knowledge that could either be public – accessible by all interacting agents or private – accessible by only selected agents (i.e. K); and a description of their current progress in pursuing the interaction, which is null at the beginning. $\Omega \overset{p}{\ni} \langle S, V \rangle$ selects an interaction model, S, from Ω, and identifies the initial set, V, of variables in S for agent p. V is instantiated with the possible set of initial value assignments as the agent p made the necessary choice, given P_g and K, which can be denoted as $c(p,P_g,K,V)$. In the expression we do not define the mechanism by which that choice is made since it varies depending on applications – anything from a fully automated choice to a decision made by a human operator.

- $\sigma(p,G_p)$ is true when goal G_p is attained by agent p.
- M is the current set of messages sent by the agents concerning the MAP to be solved. The empty set of messages is $\varnothing$.
- $i(S,M_i,V_i,S_f,V_f)$ is true when a sequence of interactions allows state S_f to be derived from S given an initial set of messages M_i and an initial list of variables V_i, and consequently producing V_f, a set of variables with solvable finite-domain constraints.
- $k_p(S)$ gives the knowledge visible to agent p contained in state S pertaining to the currently solved MAP.

$$S \overset{s}{\supseteq} S_p \leftrightarrow \exists R, D.(S_p \in S \wedge S_p = a(R,p) :: D) \tag{8}$$

- $S \overset{s}{\supseteq} S_p$ selects the state, S_p, concerning agent p, from the interaction state S. Given that in LCC, the state of the interaction is always expressed as a term of the form $m(P_s,P_g,K)$, the selection of the current state for an agent, p, simply requires the selection of the appropriate clause, $a(R,p)::D$, defining (in D) the interaction state for p when performing role R.
- $apply\_ranges(V_i,S_p,S'_p)$ is a relation that applies the currently constrained variables of V_i in agent's interaction state S_p to give an agent's constrained state S'_p.

- $S'_p \xrightarrow{M_i,S,M_n} S''_p$ is a transition of the state of agent p from S'_p to S''_p provided that the current set of inter-agent messages, M_i and any imposed finite-domain constraints pertaining to the variables currently discussed at the inter-agent level are solvable, and producing a new set of messages M_n.
- *update_ranges*(S''_p, V_i, V_n) is a relation that identifies each variable in V_i that has successfully been constrained in the new agent state S''_p and adds the newly constrained variables to produce V_n.

$$S_p \overset{s}{\cup} S \leftrightarrow (a(R,p) :: D) \overset{s}{\cup} S = (S - \{(a(R,p) :: D')\}) \cup \{(a(R,p) :: D)\} \qquad (9)$$

- $S_p \overset{s}{\cup} S$ merges the state S_p, concerning agent p, with interaction state S. The interaction state S, is a term of the form $m(P_s,P_g,K)$ and the state relevant to an individual agent S_p is always a LCC clause of the form $a(R,p)::D$. Merging S_p with S therefore is done simply by replacing in S the (now obsolete) clause in which p plays role R with its extended version S_p.
- Common knowledge in LCC, as described in section 2.2.2.1, is maintained in K, which is part of the interaction state $m(P_s,P_g,K)$. $k_p(S) \vdash G_p$ indicates that the satisfaction of an agent's goal, G_p, is derivable from K or through the agent's own internal constraint satisfaction mechanisms. This corresponds to the satisfied relation introduced with the rewrite rules of table 2.1 in section 2.2.2.3.
- Every successful interaction satisfying $\sigma(p,G_p)$ can then be described by the following sequence of relations (obtained by expanding the '*i*' relation within expression 5 using expression 6):

$$\Omega \overset{p}{\ni} S \overset{s}{\supseteq} S_{p1} \xrightarrow{M_1,S_1,M_2} S'_{p1} \overset{s}{\cup} S_1 = S_2 \overset{s}{\supseteq} S_{p2} \xrightarrow{M_2,S_2,M_3} S'_{p2} \overset{s}{\cup} S_2 = S_3 \ldots k_{p1}(S_f) \vdash G_{p1} \qquad (10)$$

Figure 3.2 provides a general overview on the basic architecture and process flow on how the formal model is enacted. As described in section 2.2.2.2, the components of the receipt message labelled as (1) in the figure include a protocol P, of the form

$P:=(T,C,K)$. Given this, set V contains the current restriction for each variable in the expanded clause of T. Once decoded, the set V is posted to the constraint store of a finite-domain constraint solver, and the rest of the message will be forwarded to the protocol expansion mechanism to determine the agent's next move in the interaction protocol as indicated in labels (2) and (3) of the figure respectively. As described in [Carlsson et al., '97; Henz and Muller, '00], the finite-domain constraint solver contains predicates that could be used for checking the consistency and entailment of finite-domain constraints, as well as solving for solution values of the variables. The domain of all variables gets narrower and narrower as more constraints are posted to the constraint store of the solver as illustrated in label (4) of the figure. If a domain becomes empty, the accumulated constraints are unsatisfied, and the current computation branch fails. At the end of a successful computation, the variables are expected to be assigned to a set of possible values that the variables can take. This set is called the current domain of the variables.

The expansion of an agent's role in a particular round of interaction requires the variables associated with the current interaction, to be instantiated with values obtained from solving the finite-domain constraints imposed by the agent on the variables, as indicated in label (5) of the figure. Successful expansion of the agent's part in the interaction protocol is determined by whether the solution values derived from solving these constraints are consistent with the existing solution values contained in set V. This allows the distinct finite-domain constraints, individually defined and solved by the interacting agents on each variable of the MAP, to be globally consistent. Once completed, an updated state of the interaction protocol, a new message content labelled as (6), and updated set V' labelled as (7), are encoded together before being handed-over to the message passing media to be retrieved by its intended recipient, as illustrated in label (8) of the figure.

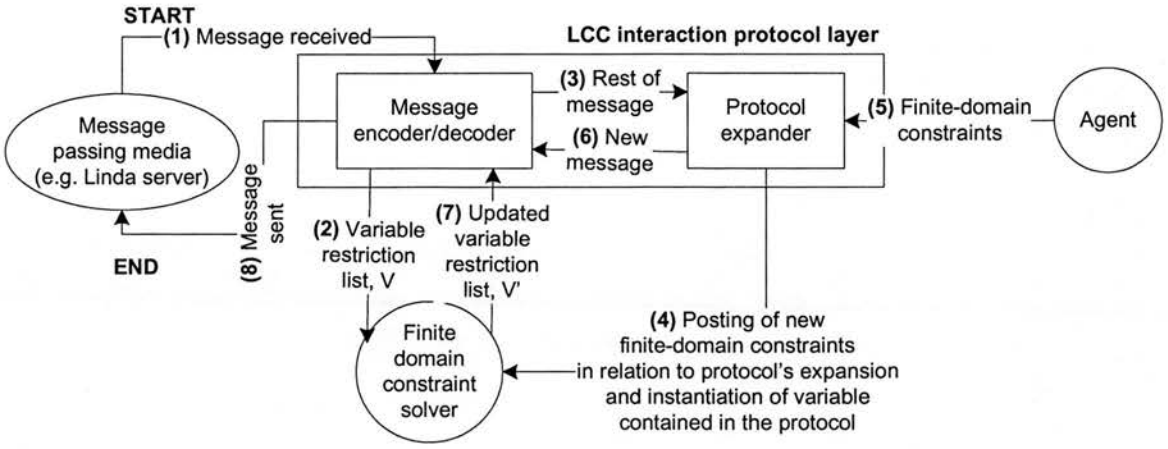


Figure 3.2: Enacting distributed constraint solving interaction in LCC

3.2 Sources of Brittleness in Interaction Protocols

As described in [Paula et al., '00], in the process of proposal exchange involving bilateral MAP solving between two agents (i.e. customer and vendor), each agent has a *private border proposal*, which is the maximum (or minimum) limit that must be respected when reaching a deal as illustrated in figure 3.3. The intersection between the agents' border proposals defines what we call the *deal range*. If the deal range is empty, then the deal is impossible. This often leads to interaction failure between the agents.

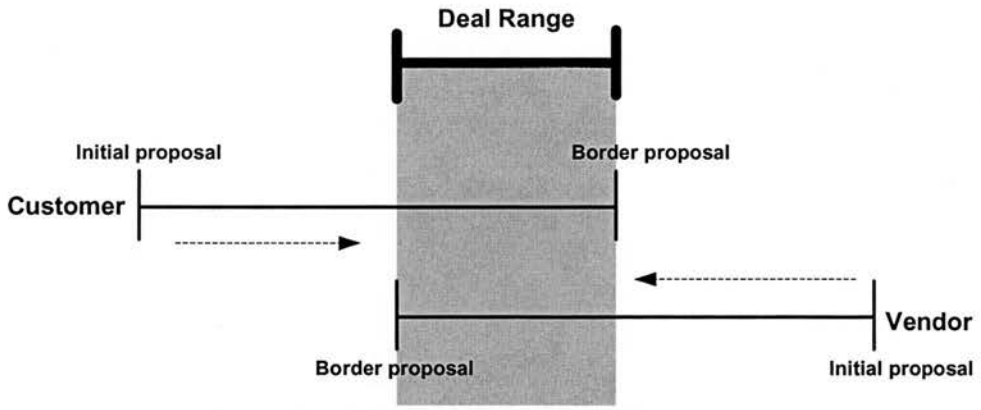


Figure 3.3: Bilateral problem solving process

An important aspect of the interaction protocol between the customer and vendor agents, as defined in table 3.1, is the message passing that communicates the attributes of the computer to be purchased. The agents will be able to continue expanding their parts in the protocol as along as the individual constraints imposed on the shared variables are satisfied. However, if this is not the case, the protocol will break as demonstrated in the following example.

We define below the knowledge private to the customer agent that includes specification on the acceptable set of values for the disk space, memory size and price attributes of the personal computer, based on the clp(FD) formalism described in section 3.1. In this example, the customer would accept a disk space of 80 Gb or 120 Gb, a memory size of 512 Mb or 1 Gb, with a total price of less than or equal to £ 300.

```
need(pc).                                                    (11)
sell(pc,s1).
acceptable(disk_space(D)) ← D in {80,120}.
acceptable(memory_size(M)) ← M in {512,1000}.
acceptable(price(_,_,P))← P #=< 300.
```

The vendor agent's local constraints are defined in the similar way as the customer. We define the available values for the attributes needed to configure a computer and relate these to its price via a simple equation (the aim being to demonstrate the principle of relating constraints rather than to have an accurate pricing policy in this example). The vendor would be able to offer disk space values of 40 Gb or 120 Gb, a memory size values of 256 Mb or 1000 Mb, with a total price that depends on the combination of a fixed base-price of £180, and the options selected for the disk space and memory size attributes.

```
attributes(pc,([disk_space(D),memory_size(M),price(D,M,P)]))  (12)
available(disk_space(D)) ← D in {40,120}.
available(memory_size(M)) ← M in {256,1000}.
available(price(D,M,P))← P #=180 +(( D div 40)*20)
                          + (( M div 256)*30).
```

As illustrated in figure 3.4, constraint graphs can be used to represent the individually defined finite-domain constraints, which are imposed on the variables of the MAP. Each node in the graph represents a variable, and each arc represents a constraint

between variables represented by the end points of the arc. Each individual graph is identified as a local constraint graph, established by the interacting agents (private to each individual agent) normally prior to the interaction (pre-interaction stage). In addition, there exist two types of solution lists (i.e. local and global solution lists). The local solution list is derived by solving the defined constraint graph at the local level. Given a solvable constraint graph, its corresponding solution list contains a set of possible solution values for the variables acquired by each individual agent. The global solution list is basically a solvable and consistent merger of the local solution values pertaining to the variables of the MAP. The global solution list is obtained as a result of solving the equality constraints of the MAP among the interacting agents. Given that in this example, the interactions among agents are handled in an issue-by-issue basis; the global list is incrementally updated with a set of satisfied solution values that depict the agreement reached by the interacting agents during the distributed problem solving process for each of the variable. In other words, the global solution list is incrementally expanded in accordance with the progression of the agents' interaction states as prescribed in the protocol.

Achieving the goal of a solvable MAP state using the protocol requires the finite-domain constraints defined in the distinct constraint graphs, and held individually by each interacting agent, lead to solution values that collectively satisfy equality constraints of the MAP. This means that the solution values for the variables, individually generated by each agent during its turn of interaction, must be globally consistent. Upon achieving this state, the variables in the dynamically expanded global solution list are assigned the set of values successfully derived from the distributed constraint solving process.

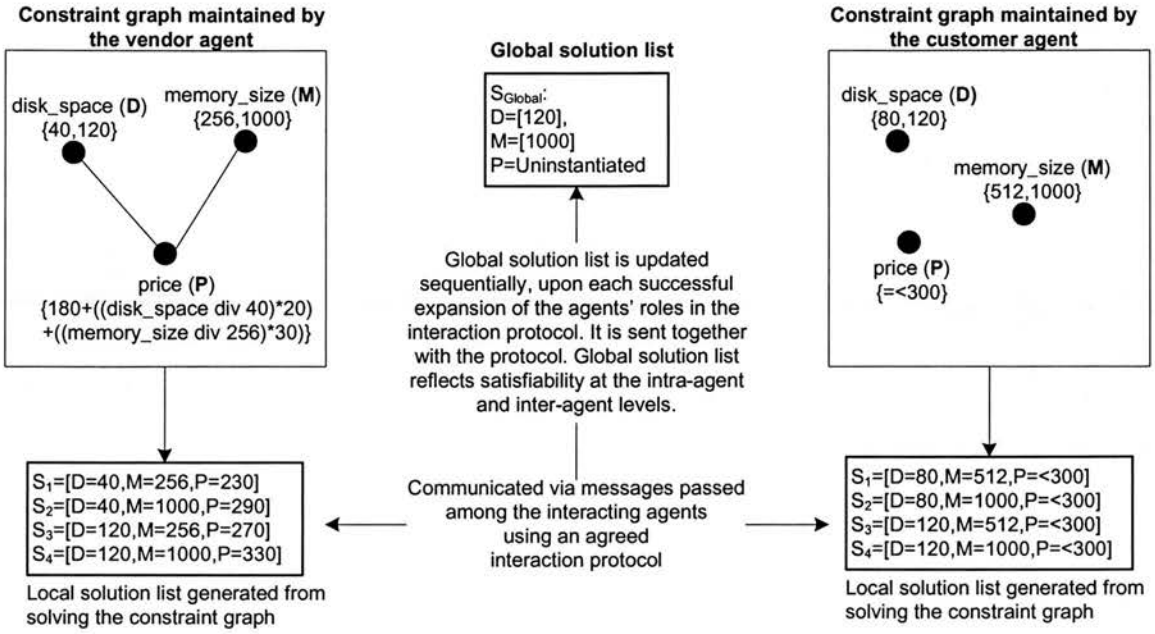


Figure 3.4: Conceptual overview of constraint graphs and solution lists involved in agents' interactions

The constraint store of the finite-domain constraint solver used by the interacting agents to provide computation on the expansion of the global solution list is ephemeral – lasting only from the period an agent receives a message until it completes its part in a particular interaction session. Thus, each time an agent reacts to a received message that places a requirement on the agent to satisfy equality constraints on the variables of the MAP, the global solution list attached together with the interaction protocol needs to be repacked and updated. During an agent's turn of posting and satisfying its part of the equality constraints to the constraint store, it may fail to maintain the consistency of the global solution list. The occurrence of this failure will prevent the collective constraint solving effort by the interacting agents from achieving a solvable MAP, as prescribed in the protocol. This over-constrained problem will cause the protocol to break, which is described in detail in the remainder of this section.

The problem is over-constrained as the possible values for the price attribute of both agents are in conflict with each other. The sequence of message passing that follows from the protocol expressions is shown in table 3.2. The interaction is between the customer, *b1*, and a vendor, *s1*. Each illocution shows a numeric illocution identifier for reference (i.e. *1..n*); the role of the agent sending the message; the message itself; the role of agent

to which the message is sent; the variable restrictions applying to the message (the term *r(V,S)* relating the possible set of solution values *S*, for the variable *V*, derived from satisfying the individually defined finite-domain constraints imposed on the variable). In specifying the solution values *S*, we follow the clp(FD) notation where:

- $[[X_1|X_n]]$ indicates an inclusive range of continuous values from X_1 to X_n;
- [[inf|X]] indicates an inclusive range of continuous values from a lower constant infinity, inf, to X;
- [[X|sup]] indicates an inclusive range of continuous values from X to an upper constant infinity, sup; and
- $[[X_1|X_1]]$ indicates a discrete value X_1. For a set of discrete values, $X_1 \ldots, X_n$, it is represented as $[[X_1|X_1],\ldots,[X_n|X_n]]$.

The first illocution is the customer making initial contact with the vendor. Illocutions two to five then are offers of possible values for the disk space and memory size attributes, each of which are accepted by the customer as they can be satisfied given the customer's intra-agent constraints. The restrictions in illocution five of [*r(M,*[[1000|1000]]*), r(D,*[[120|120]]*)*] reflect the solution values pertaining to the memory size and disk space attributes currently agreed by the agents.

No: *1*
Sender: `a(customer,b1)`
Message: `ask(buy(pc))`
Recipient: `a(vendor,s1)`
Restrictions: `[]`

No: *2*
Sender: `a(neg_vend(pc,b1,[disk_space(D), memory_size(M)],s1)`
Message: `offer(disk_space(D))`
Recipient: `a(neg_cust(pc,s1,_),b1)`
Restrictions: `[r(D,[[40|40],[120|120]])]`

No: *3*
Sender: `a(neg_cust(pc,s1,[]),b1)`
Message: `accept(disk_space(D))`
Recipient: `a(neg_vend(pc,b1,_),s1)`
Restrictions: `[r(D,[[120|120]])]`

No: *4*
Sender: `a(neg_vend(pc,b1,[memory_size(M) price(D,M,P)]),s1)`
Message: `offer(monitor_size(M))`
Recipient: `a(neg_cust(pc,s1,_),b1)`
Restrictions: `[r(M,[[256|256],[1000|1000]]), r(D,[[120|120]])]`

Table 3.2: Sequence of message passing

No: *5*
Sender: `a(neg_cust(pc,s1,[att(disk_space(D))]),b1)`
Message: `accept(memory_size(M))`
Recipient:
`a(neg_vend(pc,b1,_),s1)`
Restrictions:
`[r(M,[[1000|1000]]),`
`r(D,[[120|120]])]`
	No: *6*
	Sender: `a(neg_vend(pc,b1,[price(D,M,P)]),s1)`
	Message: `offer(price(D,M,P))`
	Recipient:
	`a(neg_cust(pc,s1,_),b1)`
	Restrictions:
	`[r(P,[[330|330]]),`
	`r(M,[[1000|1000]]),`
	`r(D,[[120|120]])]`

Table 3.2: Sequence of message passing (continued)

Given the restrictions imposed on memory size and disk space attributes in illocution five, the only offer available to be made by the vendor agent pertaining to the price attribute is *r(P,*[[330|330]]*)*, as indicated in illocution six. However, this offer is in conflict with the local solution value of *r(P,*[[inf|300]]*)*, imposed by the customer. This causes a failure of the customer to expand the interaction protocol received with the message.

Recall that the means used by each agent to maintain an appropriate role during the interaction is by expanding the clause it selects for its initial role (see section 2.2.2.2). Figures 3.5 and 3.6 are the partially expanded clauses used by agent b1 in the role of a customer and agent s1 in the role of a vendor respectively. Note that the last part of both expanded clauses, which are within the parentheses, are still open (i.e. not enclosed by *c*) because this part of interaction between the agents is incomplete. The agents are not able to fully expand their part of the protocol due to the over-constrained problem. For the customer, this is the case given the agent is not able to satisfy the *acceptable(X)* constraint, where *X* is the price attribute, imposed on the protocol term associated with the role assumed by the agent. For the vendor, the agent will not able to expand its part of the protocol until it receives the appropriate message (i.e. *accept(X)*) from the customer.

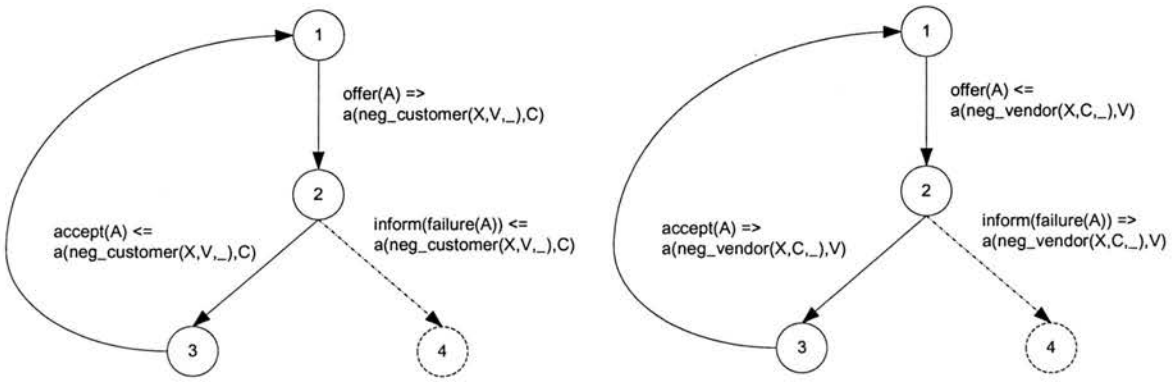
```
a(customer,b1)::
  c(ask(buy(pc))⇒a(vendor,s1)) then
  (a(neg_customer(pc,s1,[]),b1)::
    c(offer(disk_space(A))⇐a(neg_vendor(pc,b1,[disk_space(A),
                                               memory_size(B),
                                               price(A,B,180)]),s1)) then
    c(accept(disk_space(A))⇒a(neg_vendor(pc,b1,[disk_space(A),
                                                memory_size(B),
                                                price(A,B,180)]),s1)) then
    a(neg_customer(pc,s1,[att(disk_space(A))]),b1)::
      c(offer(memory_size(B))⇐a(neg_vendor(pc,b1,[memory_size(B),
                                                  price(A,B,180)]),s1)) then
      c(accept(memory_size(B))⇒a(neg_vendor(pc,b1,[memory_size(B),
                                                   price(A,B,180)]),s1)) then
      a(neg_customer(pc,s1,[att(memory_size(B)),
                            att(disk_space(A))]),b1)::
        c(offer(price(A,B,180))⇐a(neg_vendor(pc,b1,[price(A,B,180)]),s1)) then
        {accept(price(A,B,180))⇒a(neg_vendor(pc,b1,_),s1)
                ←acceptable(price(A,B,180)) then
                a(neg_customer(pc,s1,[att(memory_size(B)),
                                      att(disk_space(A))]),b1)}
```

Figure 3.5: Partially expanded interaction protocol clauses of the customer agent

```
a(vendor,s1)::
  c(ask(buy(pc))⇐a(customer,b1)) then
  a(neg_vendor(pc,b1,[disk_space(A),
                      memory_size(B),
                      price(A,B,180)]),s1))::
    c(offer(disk_space(A))⇒(a(neg_customer(pc,s1,[]),b1) then
    c(accept(disk_space(A))⇐(a(neg_customer(pc,s1,[]),b1) then
    a(neg_vendor(pc,b1,[memory_size(B),
                        price(A,B,180)]),s1))::
      c(offer(memory_size(B))⇒a(neg_customer(pc,s1,[att(disk_space(A))]),b1) then
      c(accept(memory_size(B))⇐a(neg_customer(pc,s1,[att(disk_space(A))]),b1) then
      a(neg_vendor(pc,b1,[price(A,B,180)]),s1))::
        c(offer(price(A,B,180))⇒a(neg_customer(pc,s1,[att(memory_size(B)),
                                                      att(disk_space(A))]),b1) then
        {accept(price(A,B,180))⇐a(neg_customer(pc,s1,[att(memory_size(B)),
                                                      att(disk_space(A))]),b1)
        then a(neg_vendor(pc,b1,[]),s1))}
```

Figure 3.6: Partially expanded interaction protocol clauses of the vendor agent

A quick and simple fix to the problem is through the inclusion of new clauses into the existing protocol to allow the agents to inform each other of their failure to satisfy the imposed constraints on their part of the interaction. This modification is represented in figure 3.7. In the figure, the visualisation on the possible sequence of messages sent or received when performing a role in the interaction is represented as a graph. Nodes in the graph are states in the interaction (from the perspective of the customer and vendor). Solid arcs in the figure represent clauses of the existing protocol while dashed arcs indicate the new introduced clauses.



a) Modified vendor's protocol

b) Modified customer's protocol

Figure 3.7: Interaction graphs for customer and vendor

As illustrated in figure 3.7(b), the customer's protocol is at state 2 upon receiving an offer of attribute value from the vendor. At this state, the customer can either proceed accepting this offer if it can satisfy the imposed constraints or it can inform the vendor agent of its failure to do so. For the vendor, depending on the message received from the customer, at state 2, it can either continue in its role of *neg_vendor* (i.e. state 3) or terminate the interaction (i.e. state 4) as illustrated in figure 3.7(a).

In the modified protocol, we provide the means for the agents to complete their interaction although they do not reach a solvable MAP state upon terminating the execution of the prescribed protocol. The extended protocol clauses (i.e. state 4) for both agents give an alternative to the expansion engine to continue expanding the agents' parts in the protocol if an unsatisfactory state is encountered, rather then reaching an undesirable deadlock state. The problem with this fix is that the completion of the protocol does not necessarily indicate attainment of the agents' goals pertaining to the

collective solving of the MAP. In fact, this form of corrective measure only addresses the agents' inability to complete their respective parts as specified in the protocol upon reaching an over-constrained state. It, however, does not at all address the over-constrained state of the MAP faced by the interacting agents.

The rigid feature of the protocol is inherited from the conventional constraint solving system that only allows two satisfactory states to be achieved – completely satisfied or completely violated. However, we should realise that when agents interact to solve a particular MAP, it is rarely the case that their individual constraints are completely acceptable or completely inconsistent to each other. Rather, it is normally the case that their respective constraints are partially satisfied. Therefore, given that the agents are capable to revise or relax their locally imposed constraints upon encountering an over-constrained situation while participating in the distributed constraint solving process of the MAP, the described form of corrective measure will definitely not be able to accommodate the agents' computational and interactive needs for addressing the problem. Therefore, a more sophisticated solution is required for addressing the problem.

3.3 Addressing Brittleness via Constraint Relaxation

Our approach to address this brittleness problem requires an agent to be able to adapt to the constraints on variables established by the other agents, achieved through constraint relaxation. The form of constraint relaxation considered in this work is focused on the revision of the individually assigned finite-domain constraints by a single or many agents towards the achievement of a deal.

Constraint relaxation is only possible if the agents participating in the interaction are cognitively and socially flexible to the degree they can handle (i.e. identify and fully or partially satisfy) the constraints that they are confronted with. As further emphasised in [Weib, '01], a requirement for applying efficient mechanisms for (joint) constraint relaxation and propagation is that agents are able to reason about their constraints and involve other agents in this reasoning process. This kind of reasoning must be quantitative in nature, because qualitative, purely symbolic reasoning about constraints like time and cost can be extremely complex especially in large-scale agent contexts.

More specifically, to achieve continuous flexibility through constraint relaxation an agent must be able:

i. To assign quantitative values to the constraints that express the relative importance they have to the agents;
ii. To assign quantitative values to the constraints that express the degrees to which the agent is willing to violate them;
iii. To assign quantitative values to the constraints that express the estimated risk of violating them (given the current environmental circumstances and the activity sequence the agents intend to execute); and
iv. To communicate (exchange, negotiate, refine, etc.) the quantities described in points i – iii with other agents.

Thus, for the constraint relaxation process to be accomplished, the engineering requirements expected from the interacting agents include cognitive and social requirements.

The cognitive requirement concerns the agent's internal reasoning capability that enables it to dynamically modify and redefine its own set of predefined constraints, an inherent functionality expected of agents involved in distributed constraint solving processes. This is largely provided by the mechanism to define and compute points i – iii described above. The issue of the best computational approach or constraint relaxation strategy that an agent might employ to reach to this decision is still open, and its discussion is beyond the scope of this thesis. However, a generally accepted notion is that the decision taken should be to the agent's own advantage, leading to the realisation of the eventual goal of the agent (i.e. interacting agents reaching an agreement in solving a particular MAP).

The social requirement obliges the participating agents to communicate and coordinate the constraint relaxation process with one another. This part is addressed by point iv. To achieve continuous flexibility, agents are expected to communicate in order to resolve any conflicting constraints on shared variables established by a society of agents. Therefore, the focus of the work is largely concerned with providing the agents involved with the interactive and computational mechanisms for coordinating the

relaxation of conflicting constraints. This process, expected to be handled at the protocol level is inspired by the distributed partial CSP scheme.

3.4 Overview of Constraint Relaxation Approach

The proposed constraint relaxation approach is intended to provide a mechanism for agent interaction when reconciling an over-constrained problem, and at the same time provide the necessary coordination and control to the distributed constraint relaxation tasks performed by the distinct agents at the local level. There exists a number of ways on how we could establish this approach. One possible option is to extend the current protocol so that it explicitly consists of clauses for allowing the agents to interact about revising their individual finite-domain constraints and coordinate this act whenever we anticipate an over-constrained failure might occur. However, this approach makes the existing protocol becomes unnecessary large, complicated and unwieldy. Due to this reason, it is more favourable to build it using a modular approach. This allows the existing protocol for handling agent interactions concerning the distributed constraint solving process to be maintained as it is, and the constraint relaxation protocol is developed as a new independent module. Interfacing between these two modules is only necessary when an over-constrained problem arises, as described in figure 3.8, which is an extended version of expressions 5 and 6 defined in section 3.1. These two modules are identified in the figure as S and R respectively.

The figure provides a general formal description of agent interactions over the protocol S, from the view of a single agent, p concerning a distributed problem solving process for a MAP. During the expansion of agent clauses as prescribed in the interaction model S, agent p needs to satisfy the intra-agent finite-domain constraints and inter-agent equality constraints associated with the variables, V of the MAP. However, failure to satisfy these constraints will prevent complete expansion of the clauses in the given S. As such, in this extended model, we provide a formal description on the necessary measures to interface with a constraint relaxation protocol, R, and the execution of R for addressing this problem.

$$\sigma(p, G_p) \leftrightarrow \left[\Omega \overset{p}{\ni} \langle S, V \rangle \wedge i(S, \varnothing, V, S_f, V_f) \wedge \left(k_p(S_f) \vdash G_p \right) \right] \tag{13}$$

$$i(S, M_i, V, S_f, V_f) \leftrightarrow [S = S_f] \tag{14}$$

$$\vee$$

$$\left[\begin{array}{l} S \overset{s}{\supseteq} S_p \wedge \\ \text{apply\_ranges}(V, S_p, S'_p) \wedge \\ S'_p \xrightarrow{M_i, S, M_n} S''_p \wedge \\ \left[\begin{array}{l} \text{complete}(S'_p, S''_p) \rightarrow \left[\begin{array}{l} \text{update\_ranges}(S''_p, V, V_n) \wedge \\ S''_p \overset{s}{\cup} S = S' \wedge \\ i(S', M_n, V_n, S_f, V_f) \end{array} \right] \\ \vee \\ \neg\text{complete}(S'_p, S''_p) \rightarrow \left[\begin{array}{l} i(R, \varnothing, V, R_f, V_r) \wedge \\ i(S, M_i, V_r, S_f, V_f) \end{array} \right] \end{array} \right] \end{array} \right]$$

Figure 3.8: Formal model of constraint relaxation interactions

The outcome of an expansion process is determined by comparing the agent's state prior to the expansion step (i.e. S'_p) with the state obtained after the expansion step has been completed (i.e. S''_p). The relation *complete*(S'_p,S''_p) is true if the agent's part in state S'_p is successfully expanded as reflected in state $S''p$. This allows for the process to continue updating each variable in V that has been successfully constrained in the new agent state S''_p, to produce V_n, and updating of the agent's part in S. Once this is completed, the process continues to the next state as prescribed in S. However, if the protocol's expansion resulted in a failure state, the constraint relaxation protocol R will be enacted through the $i(R,\varnothing,V,R_f,V_r)$ relation.

The relation $i(R,\varnothing,V,R_f,V_r)$ is true when a sequence of constraint relaxation processes collectively performed by the interacting agents, allows a solvable and relaxed set of variables V_r to be derived, given the following:

i. V, that specifies the state of the variables of the MAP (i.e. solution values) prior to the expansion failure of S by agent p;

ii. $\varnothing$, that specifies an initially empty set of messages. The parameter indicates the set of messages sent and received by the agents during the interactions for resolving the distributed, over-constrained MAP;

iii. R, a constraint relaxation protocol which is central to the relation, and the expansion on R resulted in R_f, that reflects the agents' progressions in the constraint relaxation process.

R provides the coordination and control to these relaxation processes, which is realised by interpreting the distributed partial CSP scheme using the LCC. Through R, a set of a possible space of constraint relaxation and interactive states for the agents involved in the relaxation process can be specified. Given that the agent p has failed to expand its part in the current prescribed interaction model of S due to an over-constrained problem, R can be viewed as a sub-protocol externally provided to p, and the other agents involved in the MAP solving, to support joint coordination and handling of locally performed constraint relaxation tasks. Once a relaxed set of variables V_r fully solvable by all the involved agents is obtained, the expansion of the agent's part in S prior to the occurrence of the described expansion failure will commence. The specification of the R component, which is interpreted from the distributed partial CSP scheme, is described in detail in chapter 4. In chapter 5, we will revisit our scenario that deals with the purchasing and configuration of a computer between the customer and vendor agents, in order to explain the detailed working of the constraint relaxation protocol, R.

3.5 Chapter Summary

In this chapter, we presented an interaction model for solving an instance of a MAP, formalised and computed using the LCC. The MAP involved a scenario that deals with the configuration and purchasing of a computer between the customer and vendor agents. Through the model, in section 3.2 we showed the impact of over-constrained problem on the agents' interactions. Based on this, we provided a discussion on the failure to achieve a solvable MAP state, brittleness of the interaction model, and a possible fix to the problem. In section 3.3, we described the means of addressing this problem using a constraint relaxation approach. We presented an overview of our constraint relaxation approach in section 3.4.

Chapter 4

Protocol Specification

As described in [Faratin and Klein, '01], the coordination of a conflict resolution task for a MAP among autonomous and heterogeneous agents requires the specification of the following two components. First, a protocol, or rules of interaction that coordinate the agents at an asocial level (i.e. synchronicity of messages) and social level (i.e. protocols that force the selection of a solution that satisfies some criteria). Second, the agent's strategy set, which can be specified as the preferred choices of the individual in how to i) generate solutions to the local/global problem and ii) how to evaluate proposals submitted by the other interacting agents in resolving the conflicts.

Within our proposed constraint relaxation approach, the former component is derived from the interpretation of the distributed partial CSP scheme, which encapsulates both the asocial and social levels. It provides the interacting agents with the mechanism for constraint relaxation at both the intra-agent and inter-agent stages. At the intra-agent stage, it specifies the computational behaviour that can be assumed by the agents in determining the current state of the constraint relaxation process. At the inter-agent stage, the synchronisation of message-passing behaviour among agents is established. The design and working aspects of the approach is described in detail in the remaining of the chapter. The latter component is regarded as a 'black box', defined privately by each individual designer of the agent, and is beyond the scope of this research.

Figure 4.1, is a revised version of figure 1.4, provides a general overview on how the coordination of constraint relaxation task between agents *a* and *b* could fit into the problem solving stages. This is accomplished through a constraint relaxation protocol depicted as ovals inter-connecting the agents at the interaction stage, and the protocol-regulated interactions between the agents are highlighted as dashed arrows in the figure. The constraint relaxation protocol provides synchronisation at both asocial and social levels for the agents to be involved in the relaxation process. This

allows the agents to take part in the collaborative task of relaxing their locally defined domain constraints (i.e. a set of original CSPs of all agents) in order to generate a relaxed set of solvable local problems which can satisfy the inter-agent constraints (i.e. equality constraints of the MAP). During the process, the domain constraints defined at the pre-interaction stage are expected to be revised in accordance with the private constraint relaxation strategies adopted by the individual agents. Consequently, through a successful completion of a constraint relaxation process, a solvable MAP, which consists of a set of mutually agreed solutions for each of the variables is obtained, as depicted at the post-interaction stage. Further details of this process are provided in the remainder of this chapter.

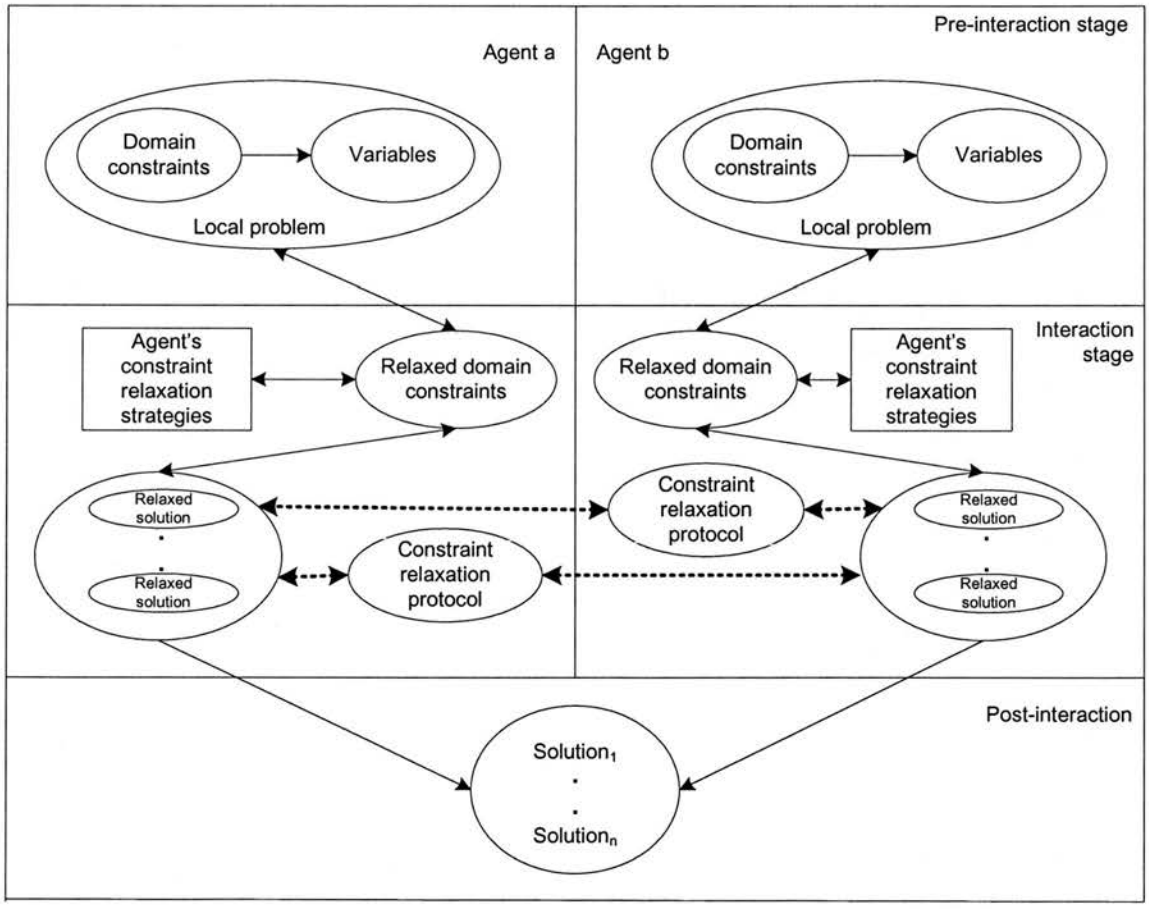


Figure 4.1: Problem solving stages and constraint relaxation task

We begin this chapter by providing a comprehensive description of how we use the distributed partial CSP scheme to implement our constraint relaxation approach. This includes a detailed discussion on the following:

1) The distance metric used (i.e. solution subset distance to compute the degree of constraint relaxation attempted by each individual agent).

2) How to find a solvable MAP among agents involved in the constraint relaxation process.

3) The global distance function for agents to compute the best constraint relaxation path to be taken.

4) Formal specification of the overall constraint relaxation process.

This is followed by an algorithm to search for a solvable MAP state with minimal distances. The algorithm also specifies how coordination among agents should take place. The chapter concludes with a description of how the constraint relaxation process is encoded into an LCC protocol. This includes a discussion of how the details pertaining to the constraint relaxation task can be tied to a set of particular agents' roles and behaviours.

4.1 Application of Distributed Partial CSP for Addressing Over-Constrained Problem

The general approach of distributed partial CSP described in section 2.5 can be specialised in many ways. This is due to the different measures of over-constrainedness based on the distinct distance metrics that could be applied between the original constrained problems and the relaxed problems. Recall that the distributed partial CSP scheme deals with crisp CSPs, with the notion of partial ordering among problems, generated in response to a series of relaxations performed on the original, over-constrained problem. If we relax problem P_1, we obtain problem P_2 that is strictly better with respect to the distance metric used. The measure of "how much" relaxation has been attempted on a problem depends on this, and the three available distance metrics are

augmentation distance, Max-CSP distance and solution subset distance. Augmentation distance counts the number of constraints values that are not shared between P_1 and P_2, while Max-CSP finds a relaxed problem that violates the minimum number of constraints. Since the first two metrics are directly concerned with constraint computation, adopting them in our constraint relaxation approach has many limitations.

First, as the agents' parts in the prescribed interaction protocol cannot be completed due to an over-constrained MAP, the required relaxation of the problem involves an autonomous application of distinct and private constraint relaxation strategies by one or more of the involved agents on their individually defined CSPs to obtain a solvable MAP. As such, adopting the augmentation distance and Max-CSP distance metrics inadvertently reveals the agents' strategies as constraint details of the local problems need to be publicly and openly shared among the agents. Second, these metrics do not reflect the actual outcome of the relaxation action performed by the agents. Max-CSP for instance is considered one-dimensional as it only takes into account one form of relaxation – the removal of conflicting constraints. Besides removing the conflicting constraints and eventually reducing the number of constraints contained in an over-constrained problem, agents can also choose other available options as described in section 2.5. These include enlarging a constraint domain to allow more solutions to be available without reducing the number of pre-defined constraints.

On the other hand, the solution subset distance is computed by looking at the cardinality of the solution sets resulted from relaxing the constraints of an over-constrained problem. By adopting this distance metric in our constraint relaxation approach, the issue with how constraints are manipulated internally by the agents in order to introduce new assignments in the solution set is no longer a concern. This means agents can fully exercise any constraint relaxation strategy that they see fit, and this information will not be revealed at the protocol level. In this metric, we are more concerned with the changes in the cardinality of solution sets due to the constraint relaxations performed by agents. Given an over-constrained MAP, we are interested in how a set of new solutions obtained from a constraint relaxation process could contribute to the achievement of a solvable MAP. Using a solution subset distance metric, a solvable

MAP state is accomplished by searching for a set of relaxed CSPs which are as close as possible to their corresponding original CSPs in terms of number of solutions.

As the focus of this research is on the finite-domain constraint problem, the solution space derivable by agents during the constraint relaxation process is guaranteed to be finite. As such, by adopting the solution subset distance metric in our approach, it enables agents to do an exhaustive search in their distributed, over-constrained problem spaces for finding solvable, relaxed CSPs, which are as close as possible to their original CSPs. This ensures that our constraint relaxation approach is complete, i.e. it eventually finds a sufficient solution or finds that there exists no such solution and terminates.

4.1.1 A Metric for Solution Subset Distance

This research specialises the solution subset distance from [Yokoo, '01] in order to address an over-constrained MAP using a distributed partial CSP scheme. Our technique finds a solvable MAP with a minimal degree of constraint relaxation, computed based on the solution subset distance metric. This is obtained when the agents participating in the constraint relaxation task generate individual problem spaces containing a set of relaxed CSPs, so that the distance between P_2, a relaxed problem selected from the set, and the original, un-relaxed problem, P_1, is within a certain bound, according to the specified distance metric. As described in the abstract distributed partial CSP model of section 2.5, the functions to provide distance computation are specified at two separate levels – local and global.

At the local level, we are mainly concerned with the computation of additional solutions introduced due to the individual relaxation attempted by the agents. This is accomplished by comparing P_2 with P_1 each time after a relaxation is performed. Given P_1, and its corresponding relaxed problem P_2, the distance metric describes how far the solutions for the two local problems are from each other. This is accomplished by associating the solutions that are already in the original, un-relaxed problem with the one introduced due to relaxation. For instance, the solution subset distance between the two comparable problems P_1 and P_2 is the number of solutions of P_2 which are not solutions of P_1. The relationship between P_1, and its relaxation, P_2, is better described using Venn

diagrams [Edwards, '04; Ruskey and Weston, '05] as illustrated in figure 4.2. In the figure, the sets S and S' respectively represent the solution sets of the original problem, P_1, and its corresponding relaxation, P_2, that is, S=sols(P_1) and S'=sols(P_2), where sols denotes the solutions to the problem. U is a universal set that represents all the possible solution values of the MAP.

The relationship described in figure 4.2(a) is a commonly addressed problem in partial CSPs and distributed partial CSPs. In this relationship, the solutions for the relaxed problem is a proper superset of those solutions for the original, that is $S' \supset S$. If S' is a proper superset of S, then the number of elements in S' is greater than the number of element in S (written as $|S'| > |S|$). Hence, there exists at least one element x of S' which is not an element of S. Though not common, other possible relationships between the original and relaxed problems are described in figures 4.2(b) and 4.2(c). For the case described in figure 4.2(b), the solution sets for the original and relaxed problems have a number of joint elements, $S \cap S' \neq \varnothing$, and also complements of each other, $S - S' \neq \varnothing$ and $S' - S \neq \varnothing$. The third possible relationship, as described in figure 4.2(c), is the case where the solution sets of both problems disjoint, that is $S \cap S' = \varnothing$. This indicates that the original problem has undergone a major relaxation process or more likely an extreme revision which produces a totally different solution set.

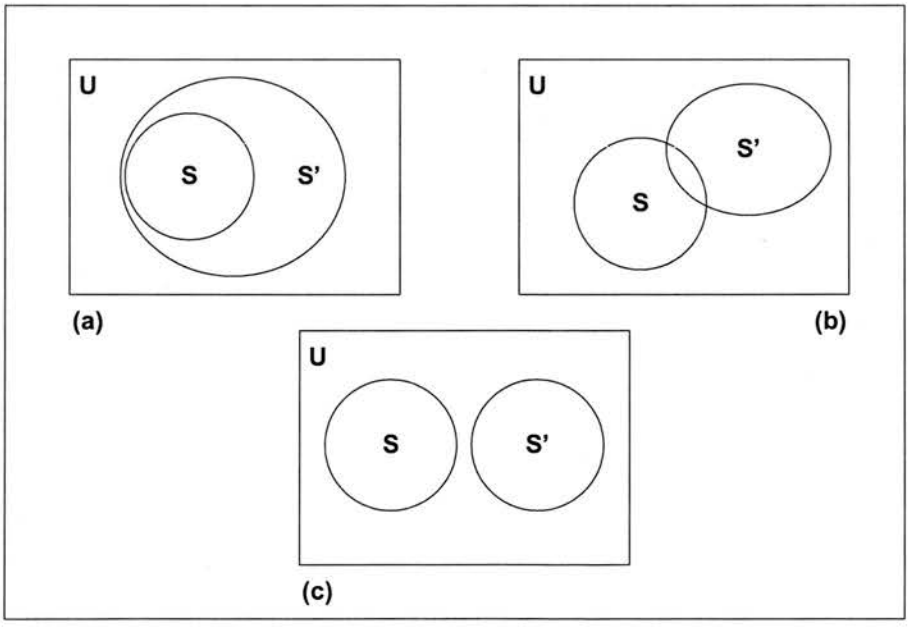


Figure 4.2: Possible relationships between the solution sets of the original(S) and the relaxed (S') problems

In order to describe the last two cases, the over-constrained scenario involving a customer and vendor agents in section 3.2 is referred. Assuming that a soluble agreement is achieved by the vendor agent relaxing its pricing policy to meet the customer's request for a lower price, then, such relaxation act will let all possible solution sets derivable from the original problem based on the initial restricted price plan to become invalid Working example on this is provided in section 5.2.

Given the three possible relationship patterns described in figure 4.2, computation of distance between the two sets (i.e. S and S') not only needs to consider the new additional solutions introduced, but also the existing solutions of the original problem that might be eliminated due to the performed constraint relaxation. Therefore, the equations in figure 4.3 describe how this is computed, where L is the union of these two components, and the distance, d, is then measured as the cardinality of S.

$$L = (S - S') \cup (S' - S) \quad (1)$$

$$d = |L| \quad (2)$$

Figure 4.3: Equations for distance computation

These equations are better illustrated using Venn diagram. In figure 4.4, we describe the value of d (highlighted as shaded areas), for each of the Venn diagrams of figure 4.2.

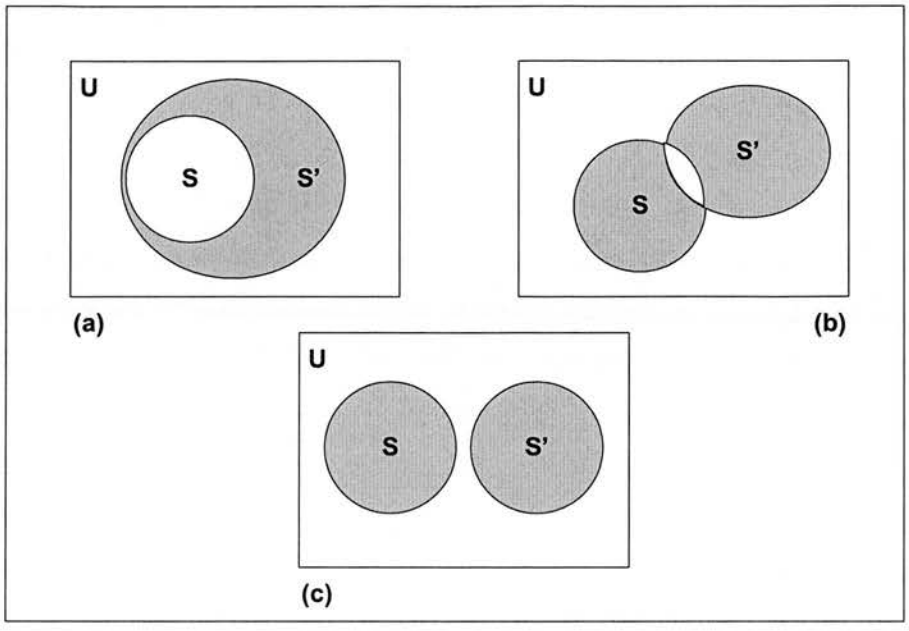


Figure 4.4: The value of 'd' derived from the solution sets of the original and relaxed problems

At the global level, we are concerned with the computation of distance of distributed problem spaces. This involves two important steps – first, finding a set of relaxed CSPs allowing for a solvable MAP state to be achieved and second, computing the global distance of the set from their corresponding original problems. Part of the first step also includes the specification of two special bounds to ensure the individual problem space generated by each agent involved in the constraint relaxation interaction is restrained. These two bounds are identified as *necessary* and *sufficient* bounds.

The disruption on agent interactions due to an over-constrained situation will normally result in a partially solvable MAP to be obtained. This MAP contains a set of fully solvable variables, assigned with solution values mutually agreed by all agents. The assignments of these variables are obtained prior to the occurrence of an over-constrained state. This is only possible if there exists a set of variables from the MAP that can be satisfied locally by each agent involved in the problem solving interaction. This set and its assigned solution values are used as the necessary bound. The necessary bound specifies that distributed problem spaces under consideration must all contain solutions that are within the bound. Assuming that all original problems individually specified by the distinct agents at the pre-interaction stage have a set of solutions which has become a

fully solvable part of the MAP, then any relaxed problems derived from the originals must contain this set of solutions, as illustrated in figure 4.5. This is necessary for preventing any relaxed problem from deviating from an already solvable part of the MAP and effectively restricts the size of the problem space under consideration. This means that any relaxed CSP obtained can only be considered if it satisfies this requirement. In the worst case scenario, this set might be empty, indicating that the interacting agents cannot reach a deal range on any of the variables of the MAP.

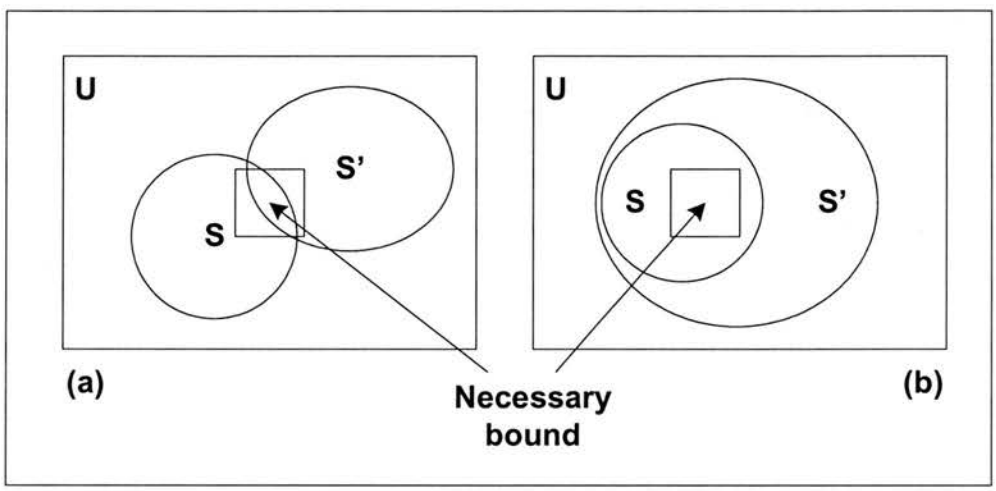


Figure 4.5: Necessary bound for restraining the relaxed problems generated by agents

A partially solvable MAP also contains a set of non-solvable variables, due to the existence of one or more agents that fail to satisfy their individual constraints concerning these variables during the problem solving interactions. This set of non-solvable variables is specified in the sufficient bound, which describes what needs to be achieved during the constraint relaxation process. Successful value assignments to the set indicates the attainment of a solvable MAP state. A set of relaxed CSPs, obtained from the problem spaces generated by the interacting agents during a particular constraint relaxation cycle, is sufficient if the additional solutions derived from these CSPs allow the initial set of non-solvable variables of the MAP to become solvable.

Both bounds give the required direction to the process of identifying locally relaxed CSPs among the agents from which a consistent, solvable MAP with an acceptable solution subset distance is derived.

4.1.2 Finding a Solvable MAP

In any MAP solving interaction through a specified protocol, there exist two possible groups of agents. Though in the actual problem solving interaction it might involve more than two agents, all the agents can be identified as belonging to either one of these two groups. The first group, *J*, consists of a set of agents that has completed its part of the protocol in solving and constraining a particular set of variables of the MAP. The second group, *K*, on the other hand, represents a set of agents whose part in the protocol is incomplete as they cannot satisfy the inter-agent or global constraints imposed on the corresponding set of variables of the MAP. Therefore, the task of finding a solvable MAP given an over-constrained distributed problem solving state, involves a series of local searches on the weakened CSPs provided by these two groups of agents during each relaxation cycle. Completion of a particular relaxation cycle can be determined when all agents have completed their roles as defined in the protocol, which is normally associated with the introduction of new solution values to accommodate the achievement of a solvable MAP state. The weakened CSPs must satisfy the necessary bound, but may not satisfy the sufficient bound. From the view of these two groups of agents who are involved in this collaborative task, the local relaxation process can be thought of as a search which starts from an initial node representing the original CSP of the agent, and follows a path until a solvable MAP state is achieved, as described in figure 4.6. The whole searching process is constrained by the specified necessary bound. The process stops when we found a combination of weakened CSPs by the individual agents that satisfy the sufficient bound with some acceptable distance between the derived solution sets. It is then said that a solvable MAP state has been achieved, and there are three possible conditions on how this is accomplished, which are described using agents *j* and *k*, instances for agent groups *J* and *K* respectively, that is $j \in J$ and $k \in K$:

1. Agent k performs the necessary constraint relaxation on its original CSP, producing a problem space containing the necessary relaxed CSPs, allowing a solvable state to be achieved without the other party, agent j, performing any relaxation on its part as illustrated in figure 4.6 (a).
2. Agent j performs the necessary constraint relaxation on its original CSP, producing a problem space containing the necessary relaxed CSPs, allowing a solvable state to be achieved without the other party, agent k, performing any relaxation on its part as illustrated in figure 4.6 (b).
3. Both agents j and k perform the necessary relaxation on their respective original CSPs, where their combined relaxation produces a corresponding set of relaxed CSPs that allow a solvable state to be achieved as illustrated in figure 4.6 (c).

However, it might also be the case that there exists no improvement towards the achievement of a solvable MAP state after a number of relaxation cycles have been performed as illustrated in figure 4.6 (d). Given this outcome, the relaxation process terminates as it simply indicates that the agents cannot reach an agreement in reconciling their differences.

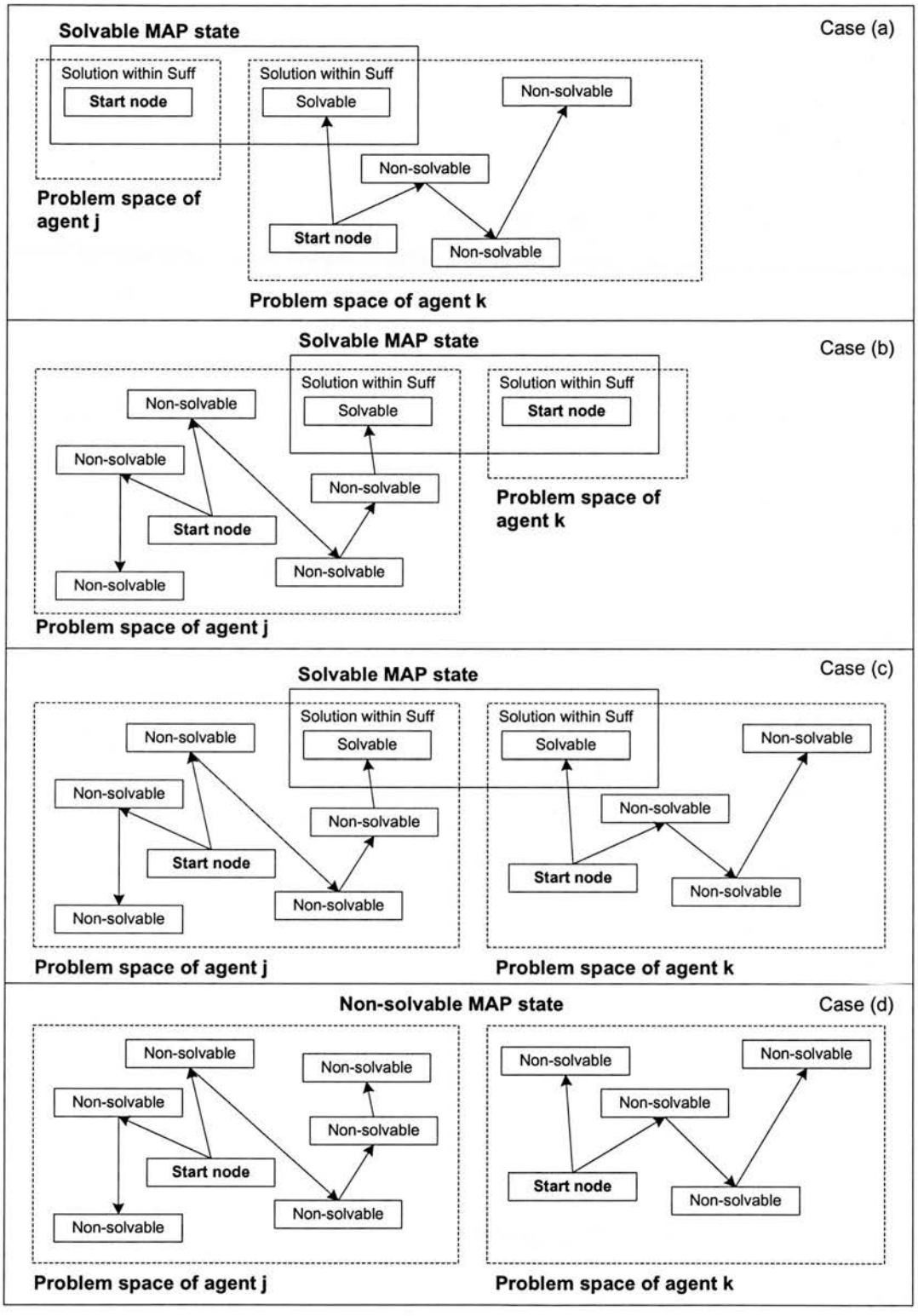


Figure 4.6: The search for a solvable MAP state

4.1.3 Global Distance Computation

In the previous sections, we respectively described the solution subset distance metric used to compute the degree of constraint relaxation attempted by each individual agent and how to find a solvable MAP among agents involved in a constraint relaxation process. The overall degree of constraint relaxation is obtained by aggregating the individual measure of distances from all of the involved agents. In this section, we describe the computation of a global distance for this purpose.

A global distance function, G, is used to measure the global distance between an original distributed CSP (i.e. a set of original CSPs of all agents) and some solvable distributed CSP (i.e. a set of solvable CSPs of all agents, generated by the agents during the constraint relaxation process) in reaching a solvable MAP state. The function can be specified as the following equation:

$$G_{Total} = \sum_{i=1}^{n} d_i \qquad (3)$$

This function provides the computation for the summation of local distances of all agents participating in the constraint relaxation task, where n is the number of agents involved in the task; d_i is the local distance for each agent i as specified in expression 2 of figure 4.3, which is the number of additional solutions introduced and existing solutions eliminated due to the relaxation individually performed by each agent on its privately defined finite-domain constraints of the MAP. We search for a combination of relaxed problems generated by the agents that minimise G_{Total}.

In order to explain a sample computation using the function, a simple example involving a relaxation process between the agents k and j are given in Table 4.1. The table provides three distinct instances of constraint relaxation cycle (i.e. 1-3) involving agents k and j, in which all produces a solvable MAP state with a different value of G_{Total}. Based on the value of G_{Total}, we can identify the best instance, which is the one with the minimum value (i.e. $G_{Total}=5$).

	d of agents		
Relaxation No.	Agent k	Agent j	G_{Total}
1.	1	5	6
2.	2	3	5
3.	5	5	10

Table 4.1: Distance metric computation for achieving a MAP solvable state

However, in some circumstances, the global distance function is inadequate to provide the necessary guidance for the selection of the best combination of relaxed problems with minimal distances over agents. For instance, table 4.2 provides a different scenario which involves three distinct instances of constraint relaxation cycle that produce a solvable MAP state. In this example, a number of solvable states are achieved with the same G_{Total} value, where the global distance function gives an equivalent rank for each instance (i.e. $G_{Total}=5$). Given this situation, a more refined global distance function is needed to provide a better comparative measure.

	d of agents		
Relaxation No.	Agent k	Agent j	G_{Total}
1.	1	4	5
2.	2	3	5
3.	5	0	5

Table 4.2: Distance metric computation for achieving a MAP solvable state

In order to address this limitation, our approach integrates the distributed maximal scheme as described in [Yokoo and Hirayama, '93; Ando et al., '03] in the computation of the global distance function. This scheme is originally intended to search for a solution that minimises the maximal number of violated constraints over agents. However, in our approach, the number of violated constraints is substituted with the solution subset distance metric. The specification of G within this approach is as follows:

$$G_{Max} = \max(d_i) \qquad (4)$$

The function provides the computation to find the maximum local distance G_{Max}, given a set of distances, d_i, for each agent i participating in the constraint relaxation task. We search for a combination of relaxed problems generated by the agents with the lowest G_{Max}.

As described by the example of table 4.2, relying solely on a global distance function, G_{Total}, might leave us with a final solution that contains a set of sizeable combinations of relaxed problems; each is equivalent in terms of distance. Therefore, in our work a hybrid global distance computation combining G_{Total} and G_{Max} is developed to perform a better search for the best combination of relaxed problems provided by the agents.

In our hybrid model, a two-stage system is employed. In the first stage, we search for a combination of relaxed problems among the agents that produces a minimal G_{Total}. For a search resulting of more than one solution, the system proceeds to the second stage. In the second stage, the G_{Max} for each remaining solution is computed and a solution with the lowest G_{Max} is selected.

Given the similar scenario as described in table 4.2, the example of table 4.3 shows a computation using both G_{Total} and G_{Max} to determine the best combination of relaxed problems to be selected. While G_{Total} gives the same rank for each instance (i.e. $G_{Total}=5$), G_{Max} identifies the combination of relaxed problems instances among agents with the lowest maximum local distance (i.e. $G_{Max}=3$).

	d of agents		Computation of solution subset distance, *G*	
Relaxation No.	Agent *k*	Agent *j*	G_{Total}	G_{Max}
1.	1	4	5	4
2.	2	3	5	3
3.	5	0	5	5

Table 4.3: Distance metric computation for achieving a MAP solvable state

4.1.4 Constraint Relaxation

The following is a detailed description of the constraint relaxation process that has been described so far:

Given a set of agents X={1,…,n} solving a particular MAP via an interaction protocol S, specified in expression 5 of section 3.1, then;

- For each agent $i \in X$, P_i is a solvable CSP defined by the agent concerning its part of the MAP at the pre-interaction stage.
- As the MAP is progressively solved by X, a set of variables, V of the MAP is incrementally instantiated with mutually agreed set of solution values, as each agent $i \in X$ propagates its P_i that is part of the MAP concerning V via S, as described in expression 6 of section 3.1. The MAP is said to be over-constrained if it consists of a set of variables, V, of which:
    - $V_S \subseteq V$, is a subset of variables that is fully solvable, in which all $i \in X$ agreed on the value assignments to V_S. That is, given $i \in X$, the value assignments to V_S is derivable from P_i. It is also possible for V_S to be empty, which means the agents cannot agree on the value assignments for any of the variable. In our work, this set of variables is specified in the necessary bound, Necs, as described in section 4.1.1.
    - $V_F \subseteq V$, is a set of variables that is partially solvable, in which given $j \in X$, the value assignments to V_F is derivable from P_j, where agent j has already completed its part as prescribed in S concerning the solving of V_F. However, there is agent $k \in X$ that cannot complete its part in S to solve V_F, as its constraints as specified in P_k concerning V_F cannot be satisfied. In our work, this set of variables is specified in the sufficient bound, Suff, as described in section 4.1.1.
- Given the over-constrained MAP, agent $k \in X$ initiates the constraint relaxation process by assuming its role as prescribed in the constraint relaxation protocol R, supplied to the interacting agents, as described in figure 3.9 of section 3.4.

- For each agent $i \in X$ involved in the constraint relaxation process, problem spaces, PS_i are made up of a number of possible weakened CSPs, generated and provided by the agents during a particular constraint relaxation cycle, in which agents relax their original CSPs (i.e. P_i) by applying constraint relaxation strategies privately held by the agents and not accessible at the interaction protocol level.
- For each agent $i \in X$, a solution subset distance metric is applied to compute the distance between each relaxed CSP, P'_i, selected from the problem space, PS_i, of agent i (i.e. $P'_i \in PS_i$), with its original, P_i, defined at the pre-interaction stage. Using this metric we identify N_i, the set of solutions not shared between the two problems, P_i and P'_i. N_i is derived by computing the union of the following two components; 1) a set of additional solutions introduced due to the selection of P'_i, and 2) a set of existing solutions of the original problem P_i, that is eliminated due to the selection of P'_i. The number of solutions identified by this union is computed as $d_i = |N_i|$, where d_i is the cardinality of N_i.
- The relaxation process involves agents $k, j \in X$ assuming their respective roles as specified in R to perform the relaxation on their P_k and P_j respectively for attaining a solvable state. A solvable state of the MAP is said to be achieved if any of the following is satisfied:
    - Agent k fully relaxes its original local problem P_k, and produces a relaxed problem, P'_k, which satisfies the necessary bound, $sols(P'_k) \supseteq sols(Necs)$. There exists at least a solution, N_k, from the set of solutions derivable from P'_k, $N_k \in sols(P'_k)$, which is consistent with the existing solutions derivable from the original local problem of agent j, $sols(P_j)$. That is, $N_k \cap sols(P_j)$. Attainment of this state indicates the satisfaction of sufficient bound, Suff. This is illustrated in figure 4.6(a). Alternatively, a similar result is achieved by agent j performing a constraint relaxation that meets the described requirements, as illustrated in figure 4.6(b).
    - Both agents $k, j \in X$ partially relax their original problems P_k and P_j respectively, and produce the respective relaxed problems P'_k, and P'_j. Both

relaxed problems satisfy the necessary bound, $sols(P'_k) \supseteq sols(Necs)$ and $sols(P'_j) \supseteq sols(Necs)$, and their combined constraint relaxations introduce new solutions N_k and N_j, where $N_k \cap N_j$. Attainment of this state indicates the satisfaction of sufficient bound, Suff. This is illustrated in figure 4.6(c).

- If no combination of relaxed problems, P'_k and P'_j, that produces a solvable MAP state is found after an exhaustive search has been performed on the problem spaces, PS_k and PS_j, of the agents k, $j \in X$ respectively, then the constraint relaxation process involving the agents k and j over the protocol R is terminated. This indicates that the agents cannot reach an agreement in reconciling their differences. This is illustrated in figure 4.6(d).

- Obtaining a solvable MAP state with the least number of constraint relaxations performed over agents k, $j \in X$ requires a search for the combinations of $P'_{k,j} \in PS_{k,j}$ which results in a solvable state to be achieved with a minimal $\sum(d_{k,j})$. Given that the search produces a number of equally ranked possible solutions, the solution with the minimal $\max(d_{k,j})$ is selected.

4.2 Algorithms for Finding Relaxed Problems that Achieve Solvable State with Minimal Distance

In this section, we describe the algorithms for finding a combination of relaxed problems provided by the agents that achieve a MAP solvable state. The state is achieve with a minimal distance from the originals among the agents. The algorithms also provide the necessary coordination for the agents to organise the constraint relaxation task. Details of the algorithms are shown in figures 4.7 – 4.10.

- The agent, *k*, who is faced with an over-constrained problem starts the algorithm by sending a **relax?** message that contains the necessary and sufficient bounds to agent $J=\{1,\ldots,n\}$ that have already constrained their part of the MAP. The sufficient bound is instantiated with the relevant solution values from the agent's original problem. These are described in figure 4.7.

- When receiving the **relax?** message, agent $j \in J$ tries to find a relaxed problem from its generated problem space that satisfies both the necessary and sufficient bounds, with the minimal distance from the agent's original problem. The set of solutions derivable from the relaxed problem, which is consistent with the sufficient bound, and its corresponding distance are returned with the **relaxed** message, if one exists. Otherwise, null values are returned. These are described in figure 4.9.
- Upon receiving all **relaxed** messages from agent J, agent k checks whether a solvable MAP state has been achieved. If it has, then the accumulated local distances, t, to reach the solvable state by this particular *relaxation_path* is computed. If the t produced by the *relaxation_path* is less than the t of the *existing_path*, then the *existing_path* is assigned with the value of the *relaxation_path*. However, if the value of t for both the *relaxation-path* and *existing_path* is equal, further computation using the maximum local distance, g, is required. The path with the minimal g is selected, if one exists. If g of both *relaxation_path* and *existing_path* is equal, then the value of *relaxation_path* is added to *existing_path*. Otherwise, no update is made on *existing_path*. These are described in figure 4.8 and the definitions for *relaxation_path* and *existing_path* are provided in figure 4.7.
- The agents will continue to the next round of relaxation cycle if the generated problem space of agent k contains relaxed problems with a distance of less then or equal to the t of the *existing_path*, and these relaxed problems have not been selected yet in any of the previous constraint relaxation cycles. The sufficient bound is revised with the solution values introduced with this selected relaxed problem and a **relaxed?** message containing the updated sufficient and necessary bounds are sent to all the other agents. These are described in figure 4.8.
- Upon completion of the constraint relaxation process among the interacting agents, the value of the *existing_path* is returned. If *existing_path* = *null*, this indicates that the agents failed to individually produce any relaxed problem that reaches a solvable MAP state. These are described in figure 4.10.

procedure **initiate** /* done by agent k for starting the algorithm */

necs; /* the necessary bound, containing variables with associated solution values agreed by all agents prior to the occurrence of an over-constrained state */

p_k; /* original problem of agent k */

suff=**sols**(p_k); /* the sufficient bound, containing all possible set of solution values for the variables of the MAP. Initially assigned with the set of solution values derived from p_k. That is **sols**(p_k) */

relaxation_path=null; /* record of achieved relaxation path for a particular constraint relaxation cycle, initially assigned to null. *relaxation_path* is in the form of $[(n,d)_i]$, where n and d are respectively the set of solutions derived by each individual agent i from the attempted constraint relaxation and the solution subset distance required by the individual agent for achieving it */

t=0; /* the summation of local distances from all agents in a particular constraint relaxation cycle to reach a solvable MAP state, initially assigned to 0 */

g=0; /* the maximum local distance selected from the list of local distances provided by all agents in a particular constraint relaxation cycle to reach a solvable MAP state, initially assigned to 0 */

existing_path=null; /* record of selected relaxation path so far, initially assigned to null. *existing_path* is in the form of $[(relaxation\_path,\ t,\ g)_r]$, where t and g are for the *relaxation_path* achieved in the constraint relaxation cycle r*/

counter =0; /* to keep track the number of receipt messages for a particular constraint relaxation cycle from each member of J agent, that is $j \in J$, so far. Initially assigned to 0 */

history_list=**sols**(p_k); /* to record the set of solution values derivable from the constraint relaxation attempted by agent for each relaxation cycle. Initially assigned with the set of solution values derived from p_k. That is **sols**(p_k) */

send (**relax?**, *suff*, *necs*) to each member of J agent, that is $j \in J$;

goto relaxation_progression mode;

Figure 4.7: Algorithm for constraint relaxation (i)

relaxation_progression mode

when agent k receives (**relaxed**, n_j, d_j) message from agent $j \in J$ **do**
  add 1 to *counter*; add (n_j,d_j) to *relaxation_path*;
  if *counter* = total number of J agent
    if $\forall(n_i,d_i)$ in *relaxation_path*, ($n_i \neq$null,$d_i \neq$null) **and** $\forall(n_i)$ is consistent, **then**
      $t = \sum d_i$; /* the accumulated local distances to reach a solvable MAP state */
      g = **max**(d_i); /* the maximum local distance to reach a solvable MAP state */
      if *existing_path* ≠ null **then**
        if $t < t$ *of existing_path* **then** assign (*relaxation_path*, t, g) to *existing_path*;
        else if $t = t$ *of existing_path* **then**
          if $g < g$ *of existing_path* **then**
            assign (*relaxation_path*, t, g) to *existing_path*;
          else if $g = g$ *of existing_path* then
            add (*relaxation_path*, t, g) to *existing_path*;
        end if; **end if**;
      else
        assign (*relaxation_path*, t, g) to *existing_path*;
    end if; **end if**;
    set *relaxation_path* to null; set *counter* to 0; let ps_k be problem space obtained from agent k;
    for all p'_k of ps_k **do**
      if sols(p'_k) ∈ *history_list* **then**
        remove p'_k from ps_k;
      else
        if $t \neq 0$ **then**
          if compute_distance(p'_k, p_k) > t **then** remove p'_k from ps_k;
          end if; **end if**; **end if**;
    end do;

    if ps_k is not empty, **then**
      for all p'_k of ps_k **do**
        select a relaxed problem, $p_{relaxed}$, from all p'_k contained in ps_k, which produces the minimal solution subset distance. That is, **compute_distance**($p_{relaxed}$, p_k) is the minimal;
      end for;
      suff=**sols**($p_{relaxed}$) ;
      add *sols*($p_{relaxed}$) *to history_list*;
      add (_ ,**compute_distance**($p_{relaxed}$, p_k)) to *relaxation_path*;
      send (**relax?**, *suff*, *necs*) to each member of J agent, that is $j \in J$;
      goto relaxation_progression mode;
    else
      goto relaxation_completion mode;
    end if;
  else
    goto relaxation_progression mode;
  end if;
end do;

Figure 4.8: Algorithm for constraint relaxation (ii)

when agent $j \in J$ received **(relax?**, *suff*, *necs*) from agent k **do**
let ps_j be problem space obtained from agent j; let p_j be original problem of agent j;
  solvable_problem=**find_solvable**(ps_j, p_j, *necs*, *suff*);
  if *solvable_problem* ≠ null **then**
    for all p' in *solvable_problem* **do**
      select a solvable problem, $p_{solvable}$, from all p' contained in *solvable_problem*, which produces the minimal solution subset distance. That is, **compute_distance**($p_{solvable}$, p_j) is minimal;
    end do;
    n_{min}= **sols**($p_{solvable}$) which is consistent with *suff*; d_{min}= **compute_distance**($p'_{solvable}$, p_j);
  else
    n_{min}= null; d_{min}= null;
  end if;
  send (**relaxed**, n_{min}, d_{min});
end do;

/**/

procedure **find_solvable** (ps_j, p_j, *necs*, *suff*)
solvable_list; /* to keep track of solvable problem contained in the problem space provided by the agents, initialised to null */
  if ps_j = null **then**
    return null;
  else
    do until ps_j is empty
      let p'_j be a problem obtained from ps_j;
      if solvable(p'_j, *suff*, *necs*) **then**
        add (p'_j) to *solvable_list*;
      end if;
      remove p'_j from ps_j;
    end do;
    return *solvable_list*;
  end if;

procedure **solvable**(p'_j, *suff*, *necs*)
  if ((**sols**(p'_j) $\supseteq$ *necs*) $\wedge$ (**sols**(p'_j) is consistent with *suff*)) **then**
    return true;
  else
    return false;

procedure **compute_distance**(p'_j, p_j)
$n_j \leftarrow$ (**sols**(p'_j) − **sols**(p_j)) $\cup$ (**sols**(p_j) − **sols**(p'_j));
$d_j \leftarrow |\, n_j \,|$;
return (d_j);

procedure **sols**(p'_j)
return all set of solutions derivable from problem p'_j;

Figure 4.9: Algorithm for constraint relaxation (iii)

```
relaxation_completion mode

  if existing_path = null then
    terminate algorithm with unsuccessful constraint relaxation;
  else
    terminate algorithm by returning existing_path value;
  end if;
```

Figure 4.10: Algorithm for constraint relaxation (iv)

Since in this research we are solely focused on the finite-domain constraint problem, the space of solution sets that the agents could derive during the constraint relaxation process of the MAP is guaranteed to be finite regardless of the constraint relaxation strategies that the agents might employ or how they specify the constraints for their individual problems at the pre-interaction stage. This ensures that our algorithms are complete, i.e. the algorithms eventually find a sufficient solution (i.e. a combination of relaxed problems that achieve a solvable state with minimal distance) or find that there exists no such solution and terminate. In the algorithms, the set of solutions obtained in a particular constraint relaxation cycle is recorded in the *history_list* to ensure that the possible combination of relaxed problems selected from the agents' problem spaces in each and every relaxation cycle are not duplicated in terms of solution sets. This means, for each distinct constraint relaxation cycle, the obtained result consists of a different set of relaxed problems from which a different solution set could be derived. The number of constraint relaxation cycle taken by the algorithms to terminate depends on the problem spaces provided by each individual agent during the constraint relaxation process. At a very minimum, it might take only a constraint relaxation cycle before termination is reached. At a very maximum, the number of constraint relaxation cycle taken by the algorithms to terminate is equivalent to the total number of possible solution sets derivable from the MAP.

4.3 Implementing Constraint Relaxation Approach in LCC

As LCC is a role-based language, it is necessary for our developed constraint relaxation approach described in detail in the previous section to be defined within the context of roles. As discussed in [Cabri et al., '02; Cabri et al., '04], there generally exist two distinct roles in any interaction protocol: that of *initiator* and that of *responder*. Both agents know when their portion of conversation is over because they had this notion of whether they initiated or responded to the conversation. For a smooth ongoing interaction between the agents participating in the constraint relaxation task, they are required to assume the designated roles as specified in the protocol. Each role in the interaction is modelled to encapsulate a set of conversation rules and behaviours applicable to the agents assuming the role. A role defines on how an agent in a given state receives a message of specified type, performs local actions, sends out messages, and switches to another state. The descriptions on the intra-agent and inter-agent interactions between the agents' major roles are given in figure 4.11, and detailed specifications with regards on how LCC is used to encode these roles and other function-specific roles expandable from these roles are given in clauses 5-13 of figures 4.12–4.14. In addition, we provide detailed definitions of the relevant parameters, which are encapsulated within the roles and passed among the interacting agents as described in clauses 14-28.

The agent faced with an over-constrained problem needs to assume the role of *initiator*, defined as clause (5) in figure 4.12, to begin the constraint relaxation process. Contained within this initial role are three major roles namely *relaxation_initiation*, *relaxation_progression* and *relaxation_completion* that reflects the stages involved in the overall constraint relaxation process. These major roles are incrementally expanded in a sequential order as illustrated by the direction of the intra-agent arrows highlighted in figure 4.11. In the *relaxation_initiation* and *relaxation_progression* agent roles, we define the following two kinds of capabilities – message passing behaviours and constraint relaxation computations. For the message-passing behaviours, we allow inter-agent interactions concerning the sending and receiving of constraint relaxation related messages between the agents to be established, maintained and coordinated. This part is

depicted as dashed arrows in figure 4.11, and detailed specifications of the behaviours are encapsulated within the *coordinator* role, defined as clause (10) in figure 4.13.

The constraint relaxation computations ensure that local actions like performing a solution subset distance given a relaxed and original CSP problems, searching for a solvable relaxed problem with a minimal distance or revising the sufficient bound after the completion of a constraint relaxation cycle, are made available and accessible to the relevant agents. This allows the involved agents to effectively participate in the constraint relaxation process of the MAP. The sets of computations, described in details in figures 4.14–4.15, are defined within two specific-function agent roles, namely *select_submit* and *ps_filteration*, identified as clause (9), and clause (11) in figure 4.13. These roles provide some ordering on the sequence of necessary actions to be performed at the local level. Eventually, the *relaxation_completion* role, defined as clause (8) in figure 4.12, marks the end of the constraint relaxation task. It allows smooth termination of the protocol that guarantees a revised set of constrained variables is properly returned if a solvable relaxed MAP state is achieved or a null value is returned if there exists none.

An agent needs to assume the role of a *responder* to become the recipient of a request message to relax its part of the over-constrained MAP. Upon receipt of the message, contained within the necessary and sufficient bounds, the *responder* assumes the *relaxation_computation* role. Within this role, the necessary computational process of finding a solvable relaxed CSP with a minimal distance given the original problem is performed. The inter-agent interactions between this role and the other roles of the *initiator* are illustrated as dashed arrows in figure 4.11. Further details with regards to the defined protocol clauses for these roles can be found in clause (12) and clause (13) of figure 4.13.

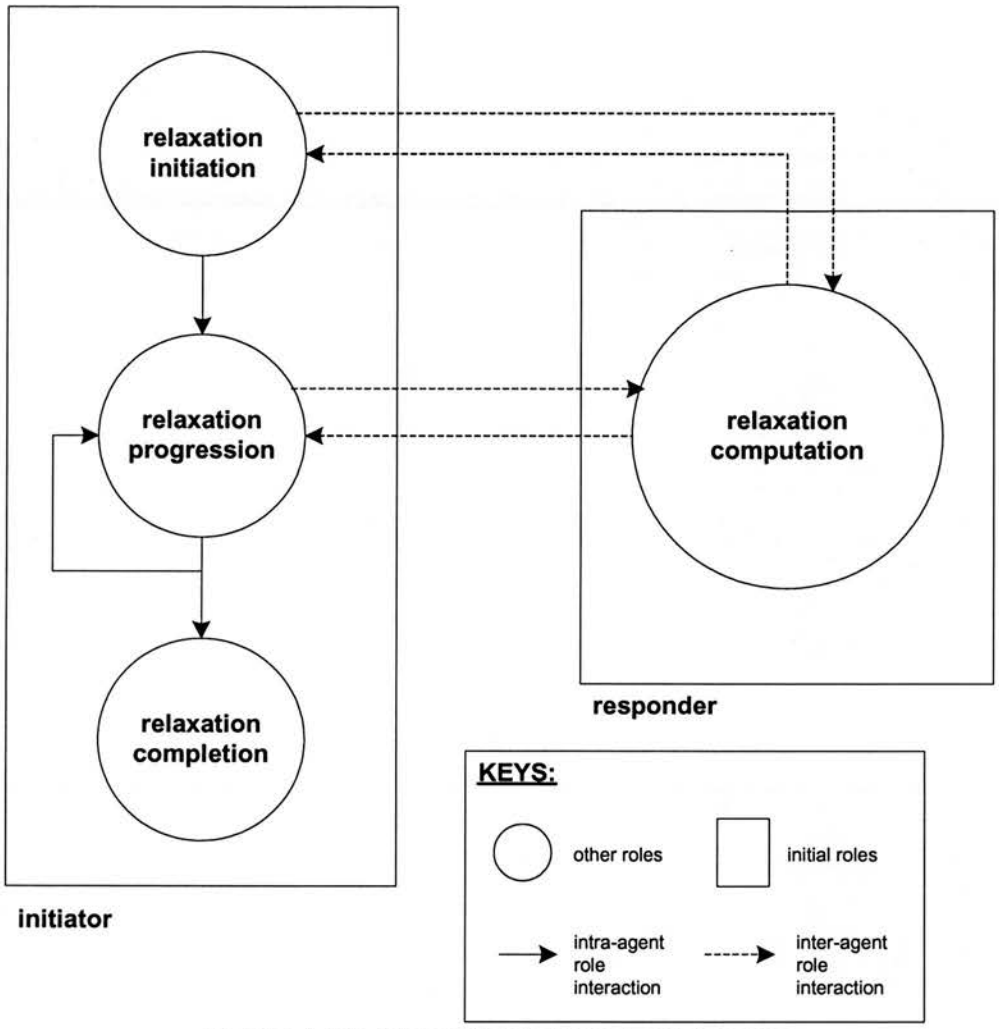


Figure 4.11: Interaction between agent roles

In LCC, the point of contact between the agents' knowledge and the defined protocol clauses of 5–13 described in figures 4.12–4.14, is provided by the following constraint clauses. These constraint clauses are associated with messages and roles of the defined protocol clauses. The knowledge to which these connections are made are obtained from two different sources:

1) **Devolved to the appropriate agent** – so that the choice of which axioms and inference procedures used to satisfy a specified constraint clause (e.g. generate a problem space consisting of relaxed CSPs) is an issue that is private and internal to the agent concerned. The constraint clauses which fall into this category are described as follows:

   - *original(O)* returns the original problem, *O*, specified by the initiator/responder at the pre-interaction stage concerning its part of the MAP, which is formalised as CSP.
   - *problem_space(PS)* returns a problem space, *PS*, consisting of relaxed problem(s) generated by the agents by applying their individual and private constraint relaxation strategies.
   - *recipient(Resp)* returns a list of agents, *Resp*, normally neighbours to the initiator, that have already completed and satisfied parts of their protocol concerning the currently solved MAP.

2) **Retained with the LCC protocol** – so that the axioms used to satisfy a specified constraint clause are visible at the same level as the protocol and the inference procedures may also be standardised and retained with the protocol (e.g. computation on distance function). The constraint clauses which fall into this category are described as follows and further details on the specification of these constraint clauses are provided in clauses 14–28 of figures 4.14–4.15.

   - *add(L1,L2,NList)* returns a list, *NList*, consisting the concatenation of two lists, *L1* and *L2*.

- *assign(Suff,P)* returns *Suff*, consisting of the set of solutions, *sols(P)* for the selected relaxed problem *P*.
- *better(NRPath,NRPath')* is true if *NRPath* is better compared to *NRPath'* in terms of distance.
- *distance(P,O,D)* returns the distance, *D*, which is the cardinality of set *U*, where *U* is the union of additional solutions introduced and existing solutions eliminated due to the constraint relaxation performed on the original problem, *O*, for obtaining the relaxed problem *P*.
- *distance_computation(TPS,O,DisTPS)* returns *DisTPS*, contained within a list of $dis(P_1,D_1),\ldots,dis(P_n,D_n)$ where for $i=1..n$, $P_i \in TPS$, that is the relaxed problems selected from the problem space, *TPS*, together with their solution subset distances, D_i, from the original, *O*.
- *find_solvable(PS,Necs,Suff,SL)* returns a set of relaxed problems, *SL*, selected from the problem space, *PS*, that satisfy the necessary bound, *Necs*, and also the sufficient bound, *Suff*.
- *g_distance(RPath,NRPath)* returns a computed distance values, *NRPath*, in the form of *gdis(RPath,T,G)* in which *T* is the total number of additional solutions and *G* is the maximum number of additional solutions, introduced over agents due to the performed relaxations, as indicated by the obtained constraint relaxation path, *RPath*.
- *g_solvable(RPath)* denotes that a globally solvable constraint relaxation state for the MAP is achieved, which is true if the obtained constraint relaxation path, *RPath*, consists a fully solvable set of solution values provided by all agents pertaining to the over-constrained variables.
- *invalid_dist_removal(DisTPS,NEPath,FPS)* returns *FPS*, contained within a set of relaxed problems, selected from *DisTPS*, that have a better or equally comparable distance if compared to the existing relaxation path, *NEPath*, obtained by the agents so far.

- *invalid_spec_removal(PS,Necs,NHList,TPS)* returns *TPS*, contained within a set of relaxed problems selected from *PS*, that generate solutions which satisfy the necessary bound, *Necs*, and do not yet exist in the history list, *NHList*.
- *locally_better(P,NEPath)* is true if the distance of a local problem *P*, is equal or better compared to the existing constraint relaxation path of the agents, *NEPath*.
- *path_computation(RPath,EPath,NEPath)* returns the constraint relaxation path, *NEPath* with a better distance, given the existing path so far, *EPath* and a newly obtained constraint relaxation path, *RPath*.
- *select_minimal(FPS,Minimal)* returns a relaxed problem, *Minimal*, selected from *FPS* that produces the most minimal distance.
- *select_path(NRPath,EPath,NEPath)* instantiates *NEPath* with the existing path, *EPath* if the existing path is better than the newly obtained *NRPath*, or otherwise *NEPath* is instantiated to *NRPath*.
- *solvable(A,Necs,Suff)* is true if the CSP formalised problem *A* produces a set of solutions that satisfy the necessary bound, *Necs* and sufficient bound, *Suff*.
- *sel({(dis$_1$(P,D),...,dis$_n$(P',D'))/min(D,...,D')})* selects a solvable problem in the form of *dis$_i$(P,D)*, where *P* is a CSP-formalised problem, and *D* is the distance. The problem with the minimal distance, *min(D)* is selected.
- *sols(P)* returns the set of solutions derivable from a CSP-formalised problem *P*.

$$a(\text{initiate}(\text{Necs}), K) ::= \tag{5}$$

$$a(\text{relax\_initiate}(\text{Necs}, O, \text{Suff}, \text{NHList}, \text{NEPath}), K) \leftarrow \begin{pmatrix} \text{original}(O) \text{ and} \\ \text{assign}(O, \text{Suff}) \text{ and} \\ \text{add}(\_, O, \text{NHList}) \end{pmatrix} \text{ then}$$

$$a(\text{relax\_progress}(\text{Necs}, \text{NSuff}, O, \text{NHList}, \text{NEPath}, \text{FHList}, \text{FEPath}), K) \text{ then}$$

$$a(\text{relax\_complete}(\text{FEPath}, \text{RelaxedPath}), K).$$

$$a(\text{relax\_initiate}(\text{Necs}, O, \text{Suff}, \text{NHList}, \text{NEPath}), K) ::= \tag{6}$$

$$a(\text{coordinator}(\text{Necs}, \text{Suff}, \text{Resp}, \text{RPath}), K) \leftarrow \text{recipient}(\text{Resp}) \text{ then}$$

$$\begin{pmatrix} \text{null} \leftarrow \begin{pmatrix} \text{g\_solvable}(\text{RPath}) \text{ and} \\ \text{path\_computation}(\text{RPath}, \_, \text{NEPath}) \end{pmatrix} \\ \text{or} \\ \text{null} \end{pmatrix}.$$

$$a(\text{relax\_progress}(\text{Necs}, \text{NSuff}, O, \text{NHList}, \text{NEPath}, \text{FHList}, \text{FEPath}), K) ::= \tag{7}$$

$$a(\text{ps\_filteration}(\text{PS}, \text{Necs}, O, \text{NHList}, \text{NEPath}, \text{FPS}), K) \leftarrow \text{problem\_space}(\text{PS}) \text{ then}$$

$$\begin{pmatrix} \begin{pmatrix} a(\text{select\_submit}(\text{FPS}, \text{NSuff}, O, \text{NHList}, \text{NEPath}, \text{THList}, \text{TEPath}), K) \\ \leftarrow \text{not}(\text{FPS} = [\,]) \text{ then} \\ a(\text{relax\_progress}(\text{Necs}, \text{TSuff}, O, \text{THList}, \text{TEPath}, \text{FHList}, \text{FEPath}), K) \end{pmatrix} \\ \text{or} \\ \begin{pmatrix} \text{null} \leftarrow \begin{pmatrix} \text{FHList} = \text{NHList} \text{ and} \\ \text{FEPath} = \text{NEPath} \end{pmatrix} \end{pmatrix} \end{pmatrix}.$$

$$a(\text{relax\_complete}(\text{FEPath}, \text{ObtainedPath}), K) ::= \tag{8}$$

$$\begin{pmatrix} \text{null} \leftarrow \begin{pmatrix} \text{not}(\text{var}(\text{FEPath})) \text{ and} \\ \text{ObtainedPath} = \text{FEPath} \end{pmatrix} \\ \text{or} \\ \text{null} \leftarrow \text{ObtainedPath} = \varnothing \end{pmatrix}.$$

Figure 4.12: Encoding of constraint relaxation as a LCC protocol (i)

$$a(\text{select\_submit}(FPS, NSuff, O, NHList, NEPath, THList, TEPath), K) ::= \quad (9)$$

$$\text{null} \leftarrow \begin{pmatrix} \text{select\_minimal}(FPS, Minimal) \text{ and} \\ \text{assign}(NSuff, Minimal) \text{ and} \\ \text{add}(HList, Minimal, THList) \end{pmatrix} \text{ then}$$

$$a(\text{coordinator}(Necs, NSuff, Resp, RPath), K) \leftarrow \text{recipient}(Resp) \text{ then}$$

$$\begin{pmatrix} \text{null} \leftarrow \begin{pmatrix} \text{g\_solvable}(RPath) \text{ and} \\ \text{path\_computation}(RPath, NEPath, TEPath) \end{pmatrix} \\ \text{or} \\ \text{null} \leftarrow TEPath = NEPath \end{pmatrix}.$$

$$a(\text{coordinator}(Necs, NSuff, Resp, RPath), K) ::= \quad (10)$$

$$\begin{pmatrix} \text{relax}(Necs, NSuff) \Rightarrow a(\text{responder}, V) \leftarrow Resp = [V \mid Vr] \text{ then} \\ RPath = [Respond \mid Rest] \leftarrow \text{relaxed}(Respond) \Leftarrow a(\text{responder}, V) \text{ then} \\ a(\text{coordinator}(Necs, NSuff, Vr, Rest), K) \end{pmatrix}$$

$$\text{or}$$

$$\begin{pmatrix} \text{null} \leftarrow \begin{pmatrix} Resp = [\,] \text{ and} \\ RPath = [\,] \end{pmatrix} \end{pmatrix}.$$

$$a(\text{ps\_filteration}(PS, Necs, O, NHList, NEPath, FPS), K) ::= \quad (11)$$

$$\text{null} \leftarrow \begin{pmatrix} \text{invalid\_spec\_removal}(PS, Necs, NHList, TPS) \text{ and} \\ \text{distance\_computation}(TPS, O, DisTPS) \end{pmatrix} \text{ then}$$

$$\begin{pmatrix} \text{null} \leftarrow \begin{pmatrix} \neg(\text{var}(NEPath)) \text{ and} \\ \text{invalid\_dist\_removal}(DisTPS, NEPath, FPS) \end{pmatrix} \\ \text{or} \\ \text{null} \leftarrow FPS = DisTPS \end{pmatrix}.$$

$$a(\text{responder}, J) ::= \quad (12)$$

$$\text{relax}(Necs, Suff) \Leftarrow a(\text{coordinator}(\_, \_, \_, \_,), K) \text{ then}$$

$$a(\text{relax\_compute}(PS, O, Necs, Suff, DL, SL, K), J) \leftarrow \begin{pmatrix} \text{problem\_space}(PS) \\ \text{and original}(O) \end{pmatrix}.$$

Figure 4.13: Encoding of constraint relaxation as a LCC protocol (ii)

$$\text{a(relax\_compute(PS, O, Necs, Suff, DL, SL, K), J)} ::= \tag{13}$$

$$\text{null} \leftarrow \begin{pmatrix} \text{find\_solvable(PS, Necs, Suff, SL) and} \\ \text{distance\_computation(SL, O, DL) and} \\ \text{select\_minimal(DL, Minimal)} \end{pmatrix} \text{ then}$$

$$\text{relaxed(N, D)} \Rightarrow \text{a(coordinator(\_, \_, \_, \_), K)} \leftarrow \begin{pmatrix} \text{Minimal} = \text{dis(P, D) and} \\ \text{N} = \text{sols(P)} \cap \text{Suff} \end{pmatrix}.$$

$$\text{add(L1, L2, NList)} \leftarrow \text{NList} = [\text{L2} \mid \text{L1}]. \tag{14}$$

$$\text{assign(Suff, P)} \vee \text{assign(Suff, dis(P, \_))} \leftarrow \text{Suff} = \text{sols(P)}. \tag{15}$$

$$\text{better(gdis(Path,T,G), gdis(Path', T', G'))} \leftarrow (\text{T} < \text{T}') \vee (\text{T} = \text{T}' \wedge \text{G} \leq \text{G}'). \tag{16}$$

$$\text{distance(A, O, D)} \leftarrow \text{U} = ((\text{sols(A)} - \text{sols(O)}) \cup (\text{sols(O)} - \text{sols(A)}) \wedge \text{D} = |\text{U}|. \tag{17}$$

$$\begin{aligned} \text{distance\_computation(TPS, O, DisTPS)} \leftarrow\ & \text{DisTPS} = (\text{DP1}, \ldots, \text{DPr}) \text{ where} \\ & \text{for i} = 1..\text{r}, \text{Pi} \in \text{TPS} \\ & \wedge\ \text{distance(Pi, O, Di)} \\ & \wedge\ \text{DPi} = \text{dis(Pi, Di)}. \end{aligned} \tag{18}$$

$$\begin{aligned} \text{find\_solvable(PS, Necs, Suff, SL)} \leftarrow\ & \text{SL} = (\text{P'}_1, \ldots, \text{P'}_t) \text{ where} \\ & \text{for i} = 1..\text{t}, \text{P'}_i \in \text{PS} \wedge \text{solvable(P'}_i, \text{Necs, Suff)}. \end{aligned} \tag{19}$$

$$\begin{aligned} \text{g\_distance(RPath, NRPath)} \leftarrow\ & \text{var(NRPath)} \\ & \wedge\ \text{T} = \sum(\forall \text{D} \in \text{RPath}) \\ & \wedge\ \text{G} = \max(\forall \text{D} \in \text{RPath}) \\ & \wedge\ \text{NRPath} = \text{gdis(RPath, T, G)}. \end{aligned} \tag{20}$$

$$\begin{aligned} \text{g\_solvable(RPath)} \leftarrow\ & \forall (\text{N, D}) \in \text{RPath}, (\text{N} \neq \varnothing, \text{D} \neq \varnothing) \text{ is True} \\ & \wedge\ \text{N is consistent with each other.} \end{aligned} \tag{21}$$

$$\begin{aligned} \text{invalid\_dist\_removal(DisTPS, NEPath, FPS)} \leftarrow\ & \text{FPS} = (\text{DisP1}, \ldots, \text{DisPq}) \text{ where} \\ & \text{for i} = 1..\text{q}, \text{DisPi} \in \text{DisTPS} \\ & \wedge\ \text{locally\_better(DisPi, NEPath)}. \end{aligned} \tag{22}$$

$$\begin{aligned} \text{invalid\_spec\_removal(PS, Necs, NHList, TPS)} \leftarrow\ & \text{TPS} = (\text{P1}, \ldots, \text{Pn}) \text{ where} \\ & \text{for i} = 1..\text{n}, \text{Pi} \in \text{PS} \\ & \wedge\ \text{Pi} \notin \text{NHList} \\ & \wedge\ \text{sols(Pi)} \supseteq \text{Necs}. \end{aligned} \tag{23}$$

Figure 4.14: Encoding of constraint relaxation as a LCC protocol (iii)

$$\text{locally\_better}(\text{dis}(P, D), \text{gdis}(\text{Path}, T, G)) \leftarrow (D < T) \vee (D = T \wedge D \leq G). \quad (24)$$

$$\text{path\_computation}(\text{RPath}, \text{EPath}, \text{NEPath}) \leftarrow \text{g\_distance}(\text{RPath}, \text{NRPath}) \wedge \text{select\_path}(\text{NRPath}, \text{EPath}, \text{NEPath}). \quad (25)$$

$$\text{select\_path}(\text{NRPath}, \text{EPath}, \text{NEPath}) \leftarrow \begin{bmatrix} \text{NEPath} = \text{NRPath} \\ \text{if better}(\text{NRPath}, \text{EPath}) \end{bmatrix} \vee \text{NEPath} = \text{EPath}. \quad (26)$$

$$\text{select\_minimal}(\text{FPS}, \text{Minimal}) \leftarrow \text{Minimal} = \text{sel}(\{\text{dis}_1(P, D), \ldots, \text{dis}_n(P', D')) \mid \min(D, \ldots, D')\}). \quad (27)$$

$$\text{solvable}(A, \text{Nesc}, \text{Suff}) \leftarrow S = \text{sols}(A) \wedge S \supseteq \text{Necs} \wedge S \cap \text{Suff}. \quad (28)$$

Figure 4.15: Encoding of constraint relaxation as a LCC protocol (vi)

4.4 Chapter Summary

In this chapter, we provided a detailed specification of the constraint relaxation protocol, which is realised from the distributed partial CSP. In section 4.1, we presented an approach, based on the distributed partial CSP, for allowing individual and distinct agents to take part in the interactive task of solving an over-constrained MAP. In the approach, the solution subset distance metric is used to compute the degree of constraint relaxation attempted by each individual agent as described in section 4.1.1. The mechanism for finding a solvable MAP among the distributed agents involved in the constraint relaxation process is described in section 4.1.2. Furthermore, we also introduced two special bounds for restraining the individual problem space generated by each agent during the constraint relaxation process; necessary and sufficient bounds. A global distance function is specified for computing the best constraint relaxation path generated by agents as described in section 4.1.3. Subsequently, a detailed description of the constraint relaxation process is provided in section 4.1.4. We showed the algorithms for finding a combination of relaxed problems that achieve a solvable state with minimal distance in section 4.2 and, in section 4.3, a detailed description on how our constraint relaxation approach is encoded into an LCC protocol is provided.

Chapter 5

Implementation and Working Example

This chapter describes the implementation aspects of our approach, followed by a discussion on the execution of the approach using the over-constrained MAP scenario of chapter 3.

5.1 Implementation

An important contribution of this thesis is not only developing the ideas of integrating distributed partial CSP with LCC, but also providing a practical and executable solution. In order to achieve this, our approach, which consists of inference procedures for performing constraint relaxation computations, needs to be implemented in a high level declarative language. As described in chapter 3, in the LCC framework, the protocol language and the expansion engine are written in SICStus Prolog [SICS, '99] and the message passing system is implemented in Linda [Carrieno and Gelernter, '89]. Therefore, we choose to implement our approach in SICStus Prolog to take advantage of the existing code for the LCC basic framework and expansion engine, and ensure smooth interfacing with these components. In addition, a finite-domain constraint solver available in SICStus Prolog (i.e. clp(FD)) is used to accommodate the computations on the solution subset distance, necessary and sufficient bounds for the set of problems contained in the agents' problem spaces.

Figure 5.1 provides a diagrammatical overview on the architecture and process flow, describing how the protocol for distributed constraint relaxation interactions is enacted in LCC. The inference procedures for performing constraint relaxation computations are defined in the constraint relaxation computational engine. The figure focuses on the execution of the protocol from the view of a single agent.

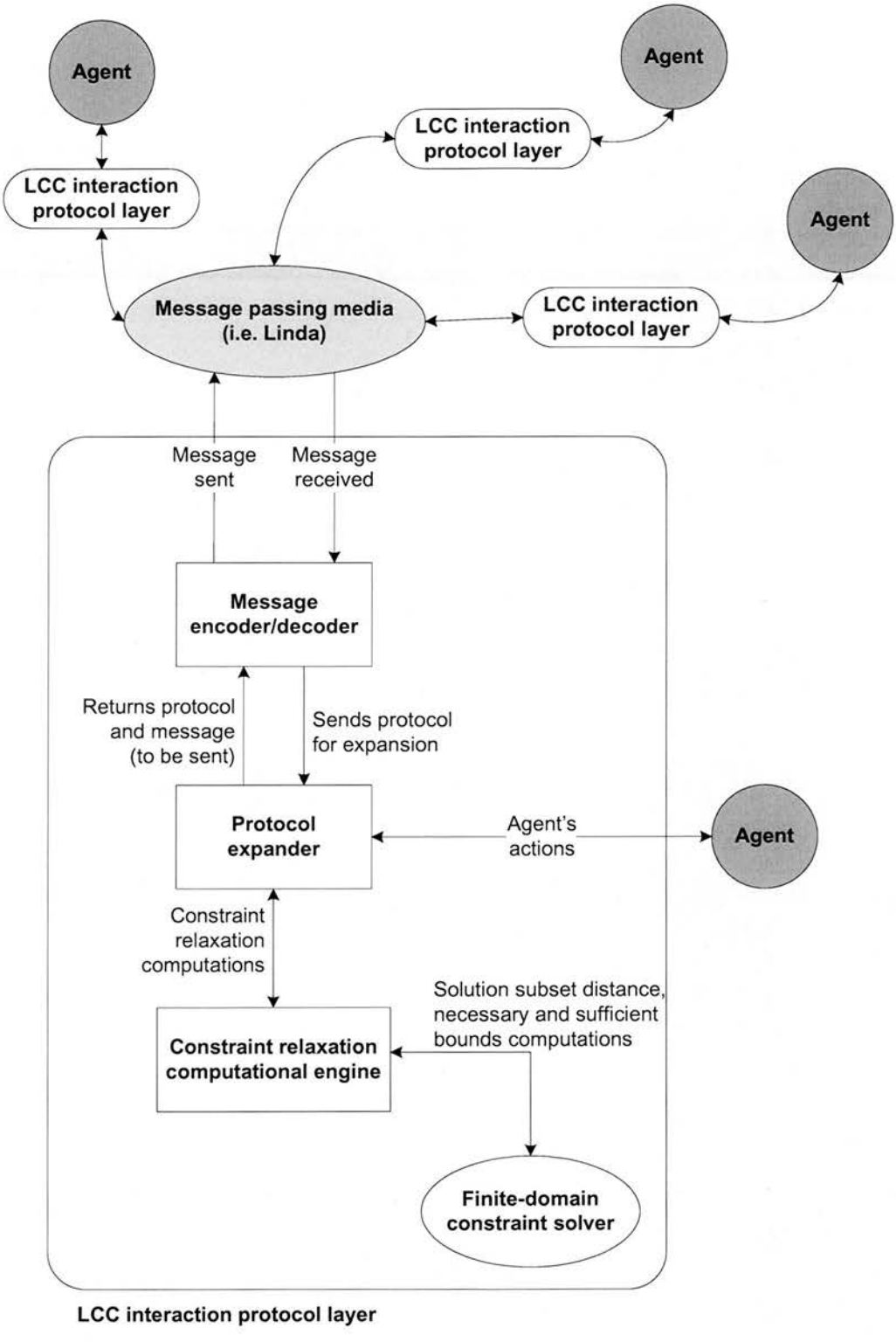


Figure 5.1: Architecture for distributed constraint relaxation interactions

Interaction via the protocol is initiated by an agent with a bootstrapping mechanism which requires a unique agent identifier, a role, and the name of a file which contains the protocol to be enacted. This will allow the file to be read and loaded into memory for the agent to use. In addition to the protocol, defined in terms of LCC agents' clauses, the file also contains a set of constraint relaxation specific knowledge pertaining to the agent (i.e. the agent's original problem and generated problem space). Once this step is completed, the agent needs to identify the appropriate clause for its role and perform the prescribed actions for that role to proceed to the next stage. This is achieved through an expansion engine that applies the set of rewrite-rules described in table 2.1 of chapter 2, onto the protocol. Each time the expansion engine finds inference procedures for performing constraint relaxation computations, a transfer of control is made to the constraint relaxation computational engine for the specific task to be executed. This process may require interfacing with a constraint solver especially for computations involving solution subset distance, necessary and sufficient bounds. If an expansion of the protocol resulted in a locution to be sent to another agent, or received from another agent, the corresponding portions of the protocol's interaction state are marked to reflect those occurring (i.e. enclosed in 'c' as described in table 2.1 of chapter 2 to indicate that the protocol clauses are already closed).

The agent clauses, the constraint relaxation knowledge base of agent, the marked agent clauses that reflect the current state of the interaction, and the locution, are merged together into a message before being sent to the Linda tuple space. A message encoder/decoder is used for receiving and transmitting messages via the tuple space. This enables messages residing in the Linda tuple space to be read, and the LCC protocol expressions contained within the messages to be extracted. The Linda tuple space uses a blackboard approach to facilitate distributed communication. In this approach, a message addressed to a specific agent as specified by the protocol is left on the space to be retrieved by the intended recipient.

The process continues with the agent checking the Linda tuple space for the messages addressed to its identifier. Once the message has been retrieved from the tuple space and decoded, the agent applies the expansion process again on the extracted LCC protocol expressions contained within the received message. The locutions received with

the message are then processed by the agent, and the agent clauses which specify the receipt of the locutions are accordingly marked to reflect the current state of the interaction. The expansion process continues for finding a suitable reply to the locutions as prescribed in the protocol, which includes satisfying any constraint attached to the agent clauses. This sequence of interaction between agents will continue until all agents have completed the expansion of their respective parts in the protocol.

For a detailed SICStus Prolog coding on these described components, please refer to appendix A.

5.2 Working Example

In order to explain a detailed expansion of the constraint relaxation protocol, we will re-visit our over-constrained scenario of chapter 3 that deals with the purchasing and configuration of a computer between a customer and vendor agents. Assuming that the universal domain values for the disk space and memory size attributes are set as D={40,80,120} Gb and M={256,512,1000} Mb respectively, then figure 5.2 and figure 5.3 provide the possible problem spaces to be obtained by the customer and vendor agents which are compatible with the necessary bound (i.e. *Necs*) of D=120 and M=1000. Contained within the problem spaces are the original CSPs (i.e. CSP_{c1} and CSP_{v1}) and the possible relaxed CSPs derived from the original CSPs (i.e. CSP_{c2}, CSP_{c3}, CSP_{c4} and CSP_{v2}, CSP_{v3}, CSP_{v4}). The forms of relaxations applied by the agents on the original CSPs are highlighted in bold, as indicated in each relaxed CSP. The individual relaxation performed on the constraints of the original problems are described as follows:

- CSP_{c2} – Enlarging of the unary constraint imposed on the disk space attribute to include a value of 40 Gb.
- CSP_{c3} – Enlarging of the unary constraint imposed on the memory size attribute to include a value of 256 Mb.
- CSP_{c4} – Enlarging of the unary constraint imposed on the price attribute from less than or equal to £300 (i.e. =< 300) to less than or equal to £350 (i.e. =<350).

- CSP_{v2} – Enlarging of the unary constraint imposed on the disk space attribute to include a value of 80 Gb.
- CSP_{v3} – Enlarging of the unary constraint imposed on the memory size attribute to include a value of 512 Mb.
- CSP_{v4} – Revising the constraint equation imposed on the price attribute where a fixed constant part of £180 is lowered to £150.

In figures 5.2 and 5.3, the set of solutions generated from these CSPs are shown in shadowed boxes, where any new solution introduced due to a performed relaxation is labelled accordingly and highlighted in bold.

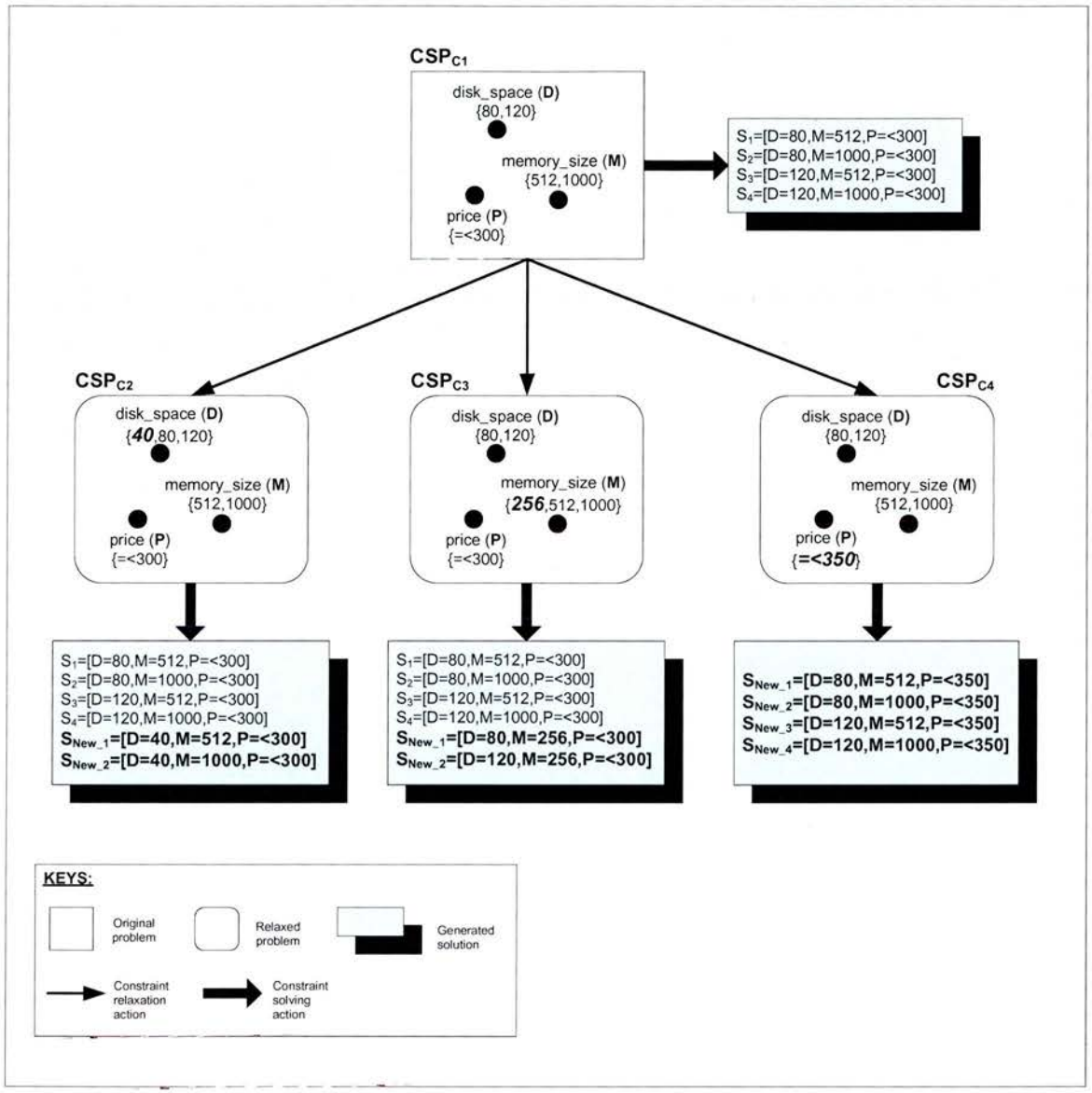


Figure 5.2: Problem space of the customer agent

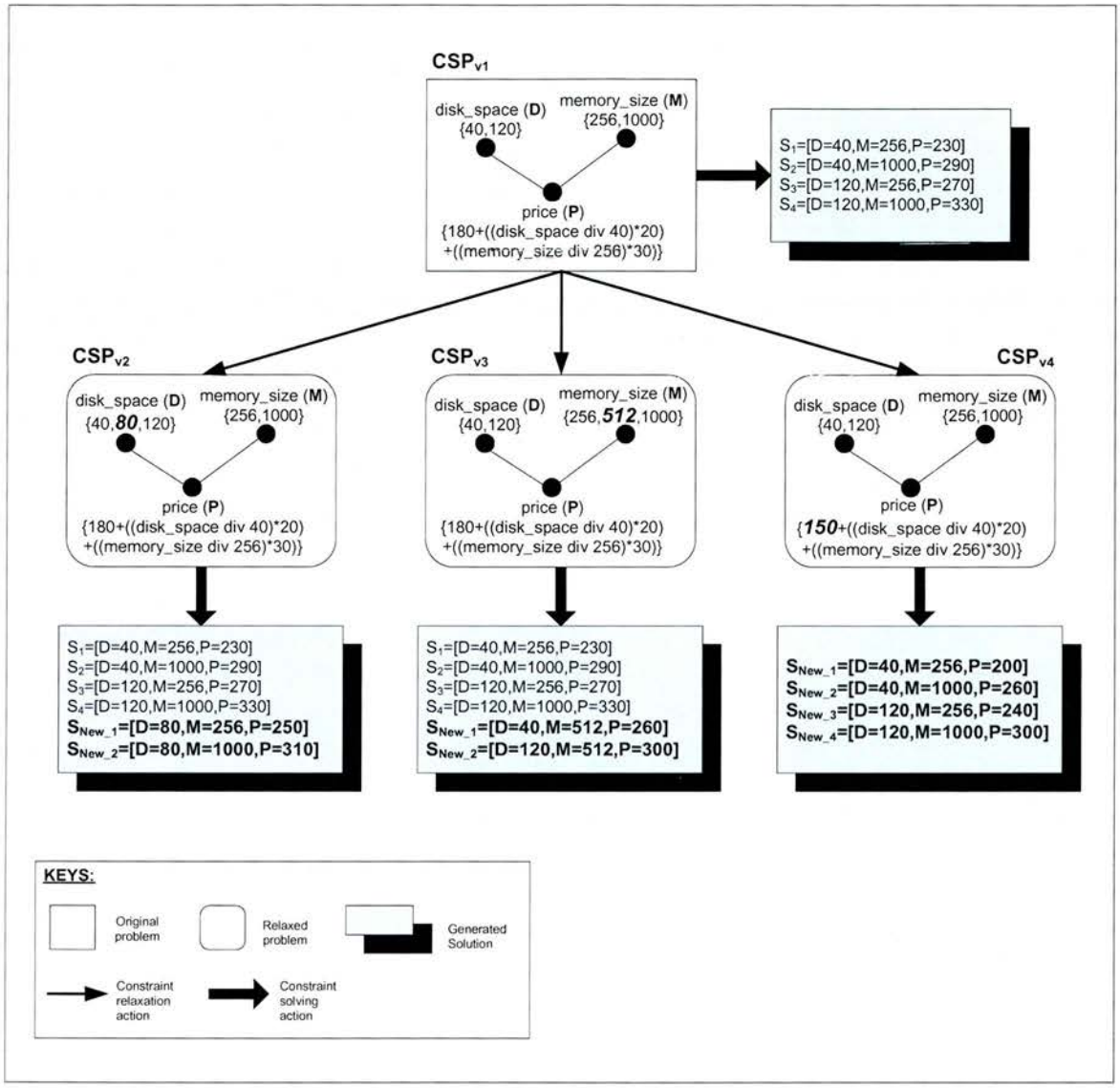


Figure 5.3: Problem space of the vendor agent

As described in section 4.3, the constraint relaxation protocol defines two kinds of capabilities to be coordinated among the interacting agents for achieving a solvable MAP state – message passing behaviours and constraint relaxation computations. The message passing behaviours are concerned with the sending and receiving of constraint relaxation related messages that follow from the protocol expressions 5–13 of figures 4.12–4.14. These are illustrated in detail in figure 5.4. The direction and flow of message passing between the customer and vendor agents are depicted using arrows. Each *relax* message sent by the customer to the vendor consists of instantiated values for the necessary (i.e.

Necs) and sufficient (i.e. *Suff*) bounds. Each *relaxed* message sent by the vendor in reply consists of a set of solution values (i.e. *N*) for allowing a solvable MAP state to be achieved among these two agents, and a solution subset distance value (i.e. *D*) on the vendor's part for obtaining this state. The term *r(V,S)* is used to indicate the possible set of solution values *S*, for the variable *V*, derived from the respective agents during the constraint relaxation process. An instance regarding the aggregation of possible solution values, S_i, for the respective set of variables under consideration, V_i, is represented by the term $s([r(V_1,S_1),\ldots,r(V_n,S_n)])$.

Constraint relaxation computations ensure local actions like searching for a solvable relaxed problem with a minimal distance or updating of parameters' values after the completion of each constraint relaxation cycle, are performed by the relevant agents as they assume their roles in the constraint relaxation interaction. These are shown in figure 5.4, where we describe the selected CSPs, and the values of *history_list*, *existing_path* and *relaxation_path* during the pre-relaxation interaction and post-relaxation interaction stages. The process of searching and selecting a CSP from the respective problem spaces of the customer and vendor agents during each constraint relaxation cycle are illustrated in figure 5.5. In the remainder of the section, we provide a detailed discussion on the execution of the protocol by the agents as illustrated in figure 5.4 and 5.5.

The customer agent, who is faced with an over-constrained problem for satisfying its part in the constraint solving interaction, starts the constraint relaxation process by assuming an *initiator* role as prescribed in the protocol. In order to begin an inter-agent interaction, the customer needs to assume a *coordinator* role, where all the message-passing behaviours for the *initiator* are specified. The customer agent begins the interaction by sending a message that contains the necessary and sufficient bounds to the vendor agent that assumes a *responder* role. The sufficient bound is instantiated with the relevant solution values from the agent's selected CSP, that is CSP_{c1} in the initial constraint relaxation cycle. Upon receipt of this message, the vendor agent expands its *responder* role to assume the *relaxation computation* role for searching a relaxed problem from its generated problem space that 1) satisfies both the necessary and sufficient bounds, with 2) the most minimal distance from the agent's original problem. As illustrated in figure 5.5, there exists two problems in the agent's problem space that meet

the first requirement; CSP_{v3} and CSP_{v4}. However, CSP_{v3} is selected since its solution subset distance of two (i.e. $D=2$) is lower than CSP_{v4} that has a distance of eight (i.e. $D=8$). Given CSP_{v3}, N is instantiated with a set of solutions derived from this CSP which matches the current solutions contained in *Suff*.

The instantiated value of N and the solution subset distance, D, of CSP_{v3}, are returned with the message sends to the customer agent. Upon receipt of this message, the global distance values, t and g, for reaching a solvable MAP state in a particular constraint relaxation cycle is computed. The global distance value t is the summation of local distances of all agents participating in the constraint relaxation task, and the global distance value g is the maximum local distance selected from the list of local distances provided by all agents. In the first cycle, the values of $t=2$ and $g=2$ are obtained. A *relaxation_path* is generated once the computation on t and g are completed. It consists of solution values, N, mutually agreed by both agents in reaching a solvable MAP state, and its associated global distance values, t and g. These are instantiated to *existing_path*, which is null at the initial stage.

A new cycle of constraint relaxation interaction will be initiated until both agents can no longer find a set of relaxed problems from their problem spaces that has a global distance which is better or equal to the global distance of the currently recorded *existing_path*. In this example, the agents are involved in three cycles of constraint relaxation interaction before a completion state is achieved. Each relaxation cycle produces a different set of relaxed problem for achieving a solvable MAP state, all with the global distance values of two (i.e. $t=2$ and $g=2$) . These are the best global distance values obtainable by the agents for solving their over-constrained MAP.

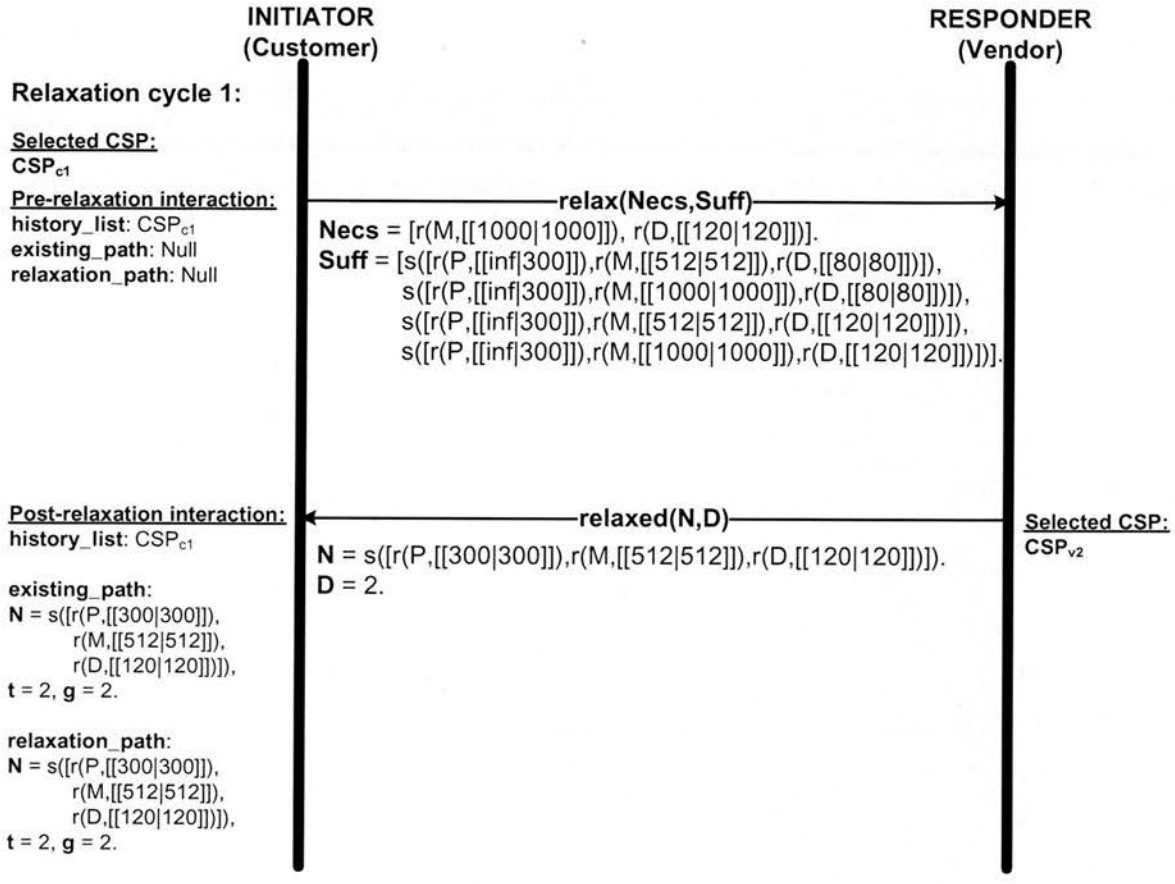


Figure 5.4: Flow of inter-agent interactions

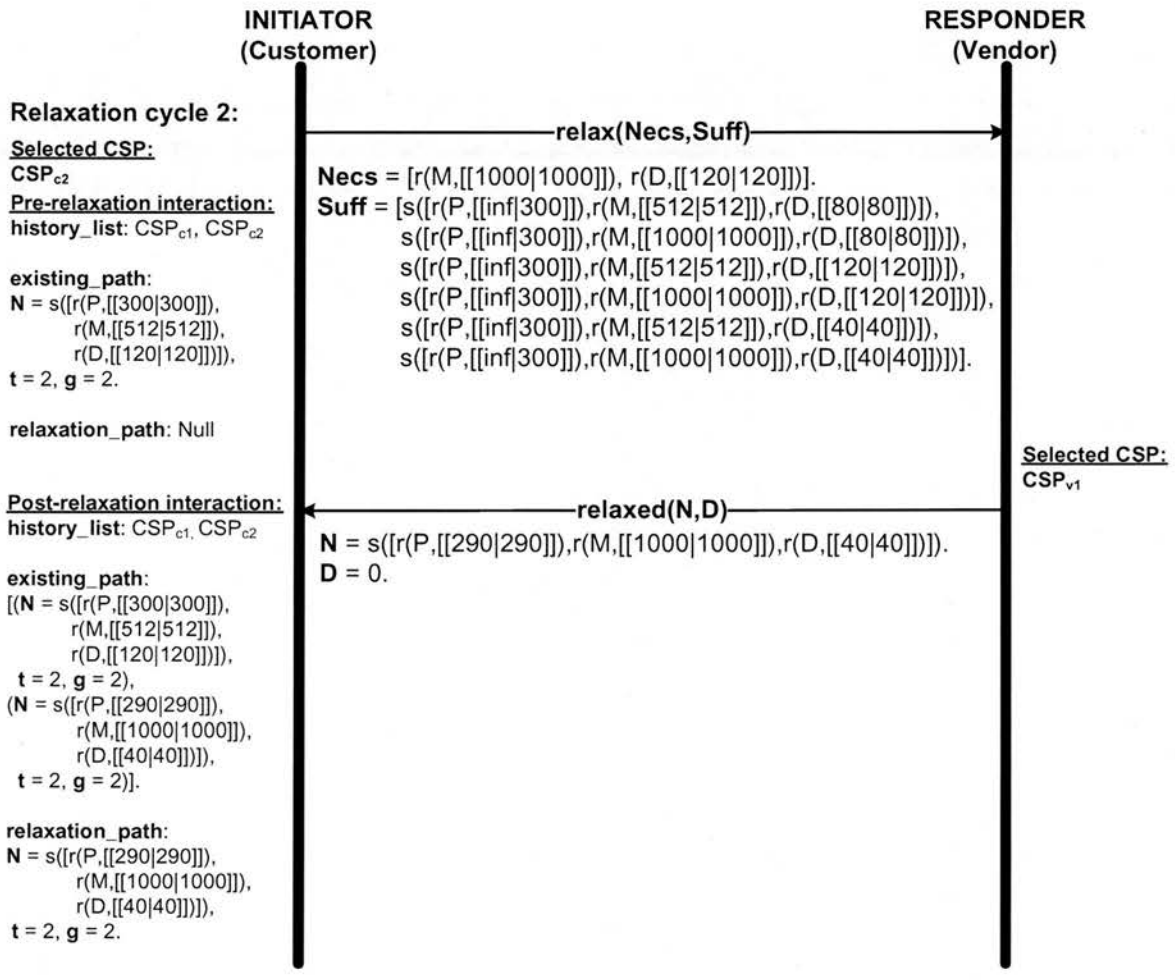


Figure 5.4: Flow of inter-agent interactions
(continuation from previous page)

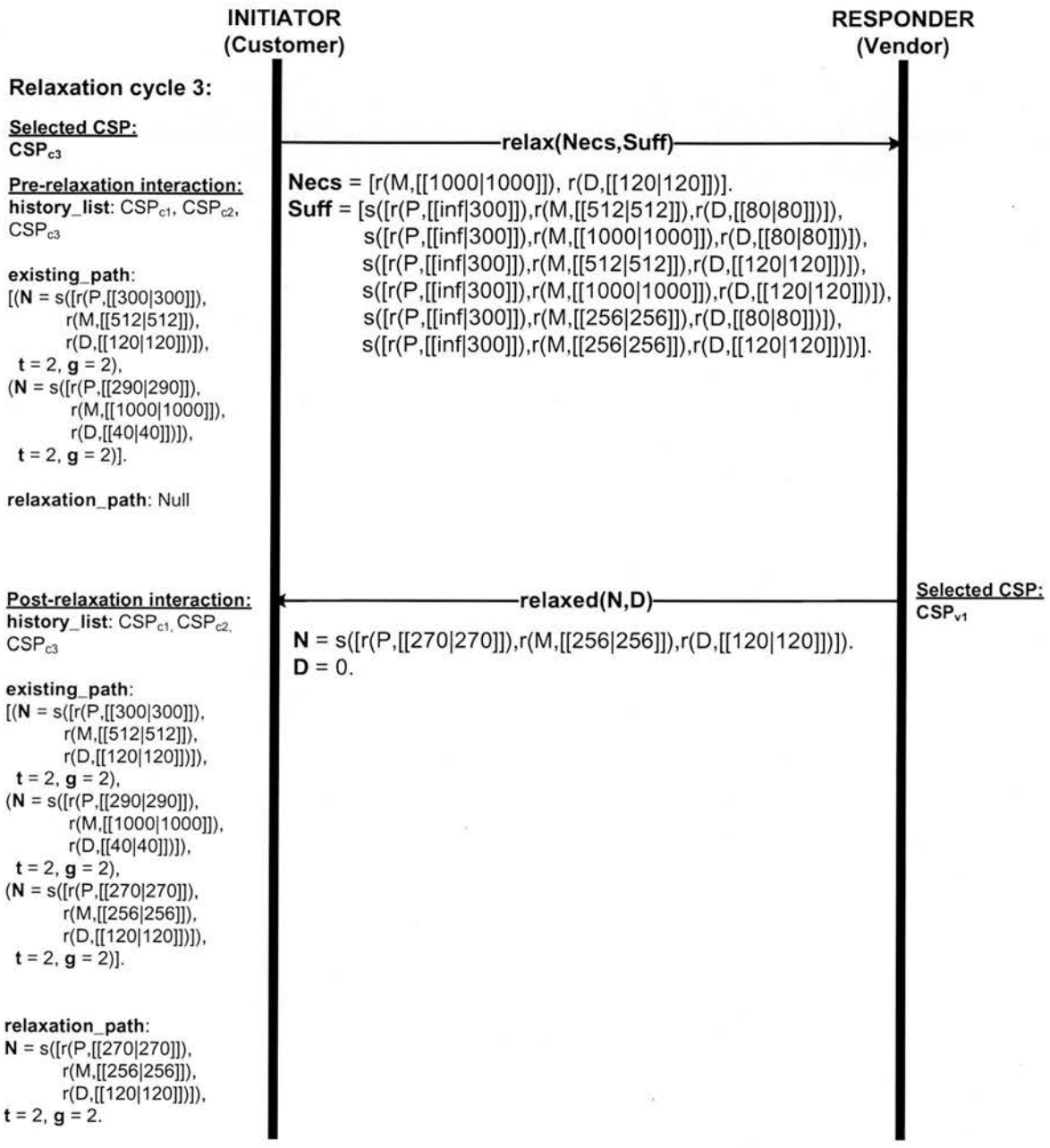


Figure 5.4: Flow of inter-agent interactions
(continuation from previous page)

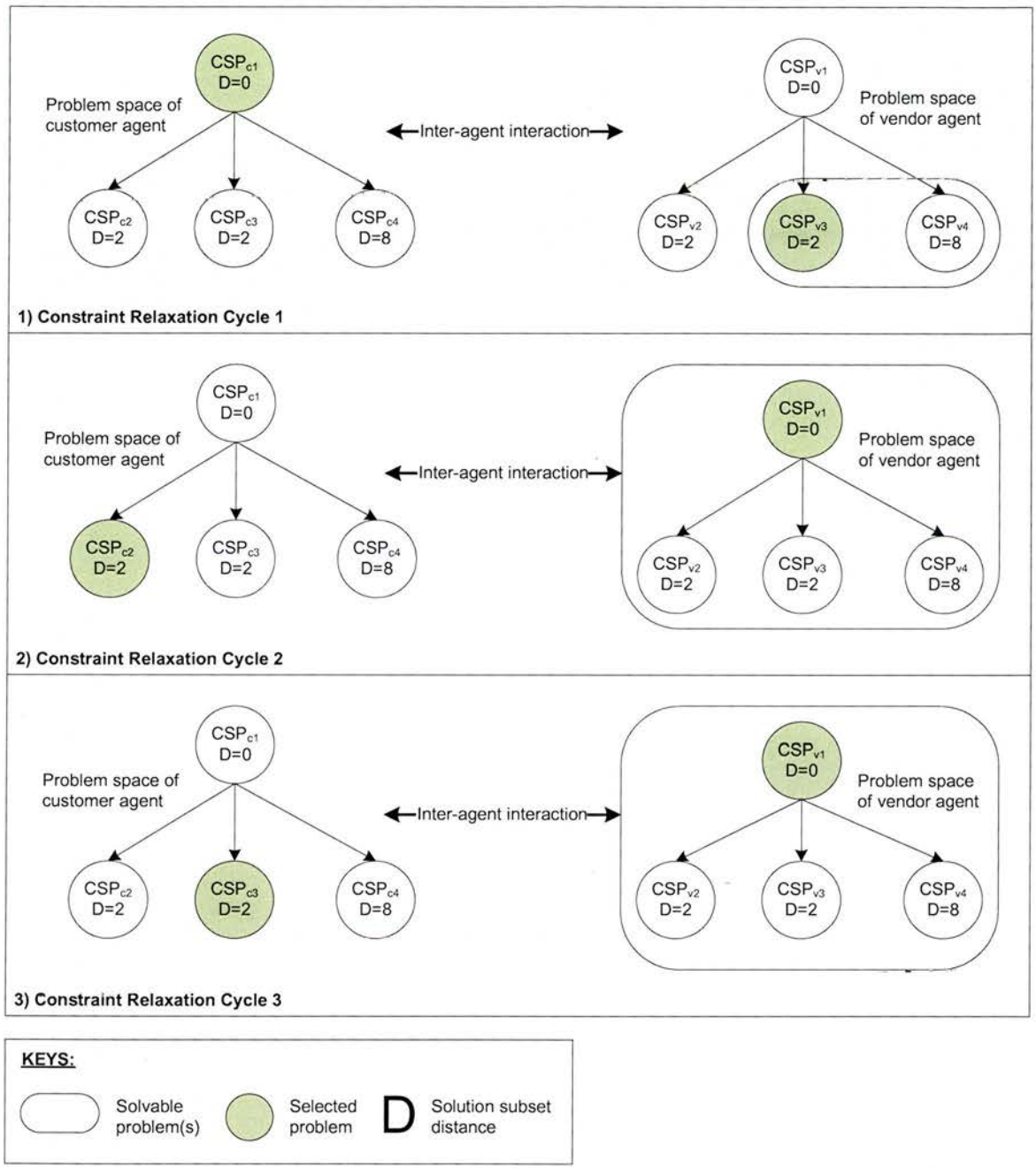


Figure 5.5: Selection of agents' CSPs during constraint relaxation computations

5.3 Chapter Summary

In this chapter, we provided a discussion on the implementation aspects of our approach. As described in section 5.1, we introduced a constraint relaxation computational engine, which consists of inference procedures for performing constraint relaxation computations. This component, which provides the necessary interface with the finite-domain constraint solver, is implemented in SICStus Prolog. Furthermore, we also described how the protocol for distributed constraint relaxation interactions is enacted. In section 5.2, we showed a detailed execution of our constraint relaxation protocol using an over-constrained scenario of chapter 3 that deals with the purchasing and configuration of a computer between customer and vendor agents.

Chapter 6

Evaluation

In this chapter we elaborate on the measures used for evaluating the constraint relaxation protocol, the set-up of the experimental test bed, the experimental results obtained from running the protocol against a set of over-constrained MAPs with different levels of hardness, and the analyses performed on these results.

6.1 Measures Used

Central to the constraint relaxation protocol is a search procedure for finding a consistent value assignment to each variable of the over-constrained MAP by all agents taking part in the process. All agents cooperate in search for a globally solvable relaxed MAP with a minimal solution subset distance. Within the distributed CSP research field, the two most common performance measurements that have been adopted to evaluate distributed search algorithms are:

1. **Time**. This measurement is motivated by the need to estimate the duration between the starting time of the algorithm and the time it returns a satisfying solution. The time performance of the algorithms has traditionally been measured in terms of computational effort, usually in the form of the number of computation cycles or steps taken by the distributed problem solvers to find a consistent solution [Davin and Modi, '05; Jung and Tambe, '05].

2. **Communication load**. Measuring the communication load poses a much simpler task, and it is generally measured by counting the total number of messages exchanged during search [Meisels, '04].

Though not perfect, the time-based measurement is a widely used method for estimating the performance of distributed search algorithms [Meisels et al., '02; Brito et al., '04], and it has also been generally accepted as a machine (and implementation) independent measure [Meisels et al., '02]. Given these considerations, we choose to adopt the time-based measurement for the purpose of evaluating the performance of our constraint relaxation protocol. As this form of evaluation method is machine independent, it is no longer necessary to run the constraint relaxation protocol in a fully distributed manner across a cluster of many computers, which is often non-trivial and impractical. Alternatively, we opt to run the protocol on a single computer using multiple threads of execution.

As the execution of the constraint relaxation protocol can be divided into a sequence of *cycles*, the time-based measurement is performed by analysing the number of cycles taken by the agents to complete their respective parts in the protocol. A *cycle* is defined as one unit of protocol progress in which all agents, in their respective roles as specified in the constraint relaxation protocol, enacted the following three behaviours:

i. Agents receive messages sent to them from the neighbouring agents to whom the equality constraints on the over-constrained MAP are shared;
ii. Agents generate the necessary problem space contained within a set of relaxed problem(s) and perform the necessary computation for finding a relaxed problem with a minimal solution subset distance;
iii. Agents send messages to the corresponding neighbouring agents together with the solvable values the meet the distance specification, if there exist one.

In a cycle-based execution as described in figure 6.1, all involved agents perform their parts as prescribed in the protocol – starting with the agent in the role of an *initiator* sending a message contained within the current *necessary* and *sufficient* bounds (i.e. *m(Necs,Suff)*) to the agent(s) assuming the role of a *responder*. Upon receipt of this message, the agent in the role of a *responder* performs local computations that include finding a relaxed problem from the locally generated problem space which satisfies the necessary and sufficient bounds, with a minimal solution subset distance. Once this is complete, a reply message contained within the agent's solution subset distance and

additional solutions derived from the selected relaxed problem (i.e. *m(Distance,Add_sols)*) which are compatible with the sufficient bound, is sent to the *initiator*, if the relaxed problem is found. Otherwise, a failure message (i.e. *m(nil,nil)*) is sent. On the *initiator* part, upon receipt of this message, a computation is performed to determine on whether a solvable MAP state with a minimal accumulated solution subset distance has been achieved, and if it does, the constraint relaxation path obtained so far is accordingly updated. An agent does not move to the next cycle until all the other agents, whom the agent is currently interacting with, have fulfilled their roles as prescribed in the protocol for a particular constraint relaxation cycle. A complete cycle is realised when each of the involved agents has completed its assigned part.

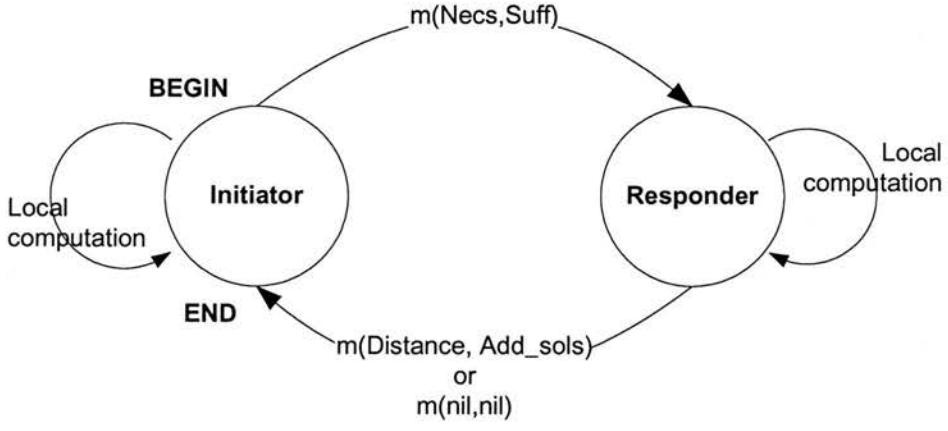


Figure 6.1: A complete relaxation cycle

The use of cycles as an evaluation metric gives a number of advantages. First, it is hardware independent. Hence, the evaluation is not affected by the different machines use in the protocol execution. Other forms of measure like the duration of time (either physical or CPU) taken to reach a solution do not necessarily corresponds to the performance evaluation of the protocol since they are dependent on the expected diverse and independent machine architecture upon which the protocol might be deployed. Second, the metric is also independent of the interaction forms held by the agents. Irrespective of the interaction method (i.e. parallel or linear) used by the agents, the number of relaxation cycles will remain the same.

6.2 Experimental Test Bed

The performance of many distributed constraint satisfaction techniques is evaluated primarily on satisfiable instances, where the biggest concern is on how they perform given a different set of constraints complexity. Within the distributed problem solving context, the focus is on the different difficulty and complexity level of the problem spaces to be solved, signified by the number of variables involved, intra-agent and inter-agent constraints. Therefore, in our experiment, we evaluate the constraint relaxation protocol using a set of over-constrained MAPs with different levels of hardness. Each MAP to be solved by the agents via the protocol is set to consist of five variables. Though the size of this problem is in an absolute sense small, it is complex enough to be representative of the normal type of problem solved by agents in a distributed environment. This size is feasible considering that the problem spaces generated by the agents during the constraint relaxation process are simulated using an exhaustive, distance-guided approach, to be described in the next section. Adding more variables will only means more works to be done as the problem spaces of the agents are expanding, but without any new insight into the observations that have already been obtained. In our case, each agent needs to provide value assignments to these multiple variables for satisfying its part in the MAP. In the experiment, each variable of the MAP is allocated a distinct set of universal domain values, which is of the same size. This is considered more difficult than the assumption normally held in many distributed problem solving related works, where one agent only handles a single variable [Yokoo, '01].

The experiment consists of two phases, a) a problem generation phase of the over-constrained MAPs and, b) a distributed constraint relaxation phase of the over-constrained MAPs via the protocol. We describe each phase in the following sub-sections.

6.2.1 Problem Generation Phase

In the problem generation phase, we provide various problem settings controlled by two important parameters – domain size and constraint compatibility. Systematic changes in these parameters generate a wide variety of problem settings, and enable us to evaluate how the protocol will perform given these different set-ups. In this work, parameter selection for the MAP is motivated by the experimental investigation in the CSP/DCSP literature, which is accordingly revised from the work reported in [Yokoo, '01; Jung and Tambe, '05] to accommodate our needs.

First, we vary the domain sizes from 4 to 7 (4, 5, 6, and 7). The purpose of this is to check the impact of having different domain sizes on the performance of the constraint relaxation protocol.

Second, we make variations in the constraint compatibility which has shown great impact on the hardness of the MAP. We distinguish external constraints from local constraints in defining the constraint tightness to analyse the effect from each class of constraints on the performance of the protocol.

Compatibility of external constraints: Compatibility of external constraints (i.e. equality constraints) is one of the primary factors that determines the hardness of over-constrained MAPs to be solved by agents via the protocol. As a MAP is composed of a set of variables, compatibility level indicates the number of variables from this set that the agents could agree on their value assignments. A low compatibility level reflects the existence of sizeable number of variables with conflicting value assignments, and vice versa.

Given a set of problems locally defined by each of the interacting agents, we require the external constraints imposed on each of the corresponding variables to be compatible for a globally solvable MAP to be derived. The compatibility of external constraints (i.e. equality constraints of the MAP) is computed based on the number of variables contained in a MAP to be solved by the agents. For instance, given that the MAP consists of five variables, the 20% compatibility level of external constraints reflects the agents' agreement only on the values of a single variable, and the 80% compatibility level of

external constraints reflects the agents' agreement on the values of four out of five variables. Given the size of our problem, we vary the compatibility level of external constraints from 20% to 80% with intervals of 20%. Note that 0% and 100% cases are not tried since the MAP doesn't have any solution for the 0% case (i.e. interaction agents cannot find an agreement on the assigned values for any of the variable) and every value assignment is a solution for 100% case (i.e. interacting agents mutually agree on the assigned values for all of the variables).

Compatibility of local constraints: As described earlier, the specification of local constraints is private. In an actual setting, given a distinct set of universal domains containing a different set of values, the agents have complete autonomy to construct appropriate local constraints for the assignment of these values on each variable of the MAP. Though the agents are autonomous to decide on constraint types and density for their parts of the MAP, for the purpose of this evaluation, we set a uniform constraint setting at the local level across agents. In the setting, the number of domain values that could be assigned to each variable is fixed based on a predetermined scale. With this form of construct, we could safely omit any factor attributed to the diverse types of constraints established by each individual agent that might influence the results obtained in the evaluation.

In assigning domain values to a variable, the number of ways of selecting *r* values from *n* distinct values contained in a universal domain can be computed using the equation specified in figure 6.2:

$$C_n^r = \frac{n!}{r!(n-r)!} \text{ for } r = 0,1,2,...,n$$

Figure 6.2: Number of combination of *r* values from *n* possible values

In combinatorial mathematics, a combination, C_n^r, is formally defined as the total number of subset of *r* values, without regard to order and without repetition, that can be selected from a set of *n* distinct values. The number of combinations equals the number

of permutations, *n!*, divided by the number of orderings. The number of ways a pool of *r* values can be ordered equals *r!*. The size of C_n^r for a discrete set of *r=0,1,2,..,n* is symmetrical as described in figure 6.3. In the case when *n* is even, the maximal size of C_n^r is at $r = \left\lceil \frac{n}{2} \right\rceil$, however, when *n* is odd, the maximal size of C_n^r is at $r = \left\lfloor \frac{n}{2} \right\rfloor$ and $r = \left\lceil \frac{n}{2} \right\rceil$.

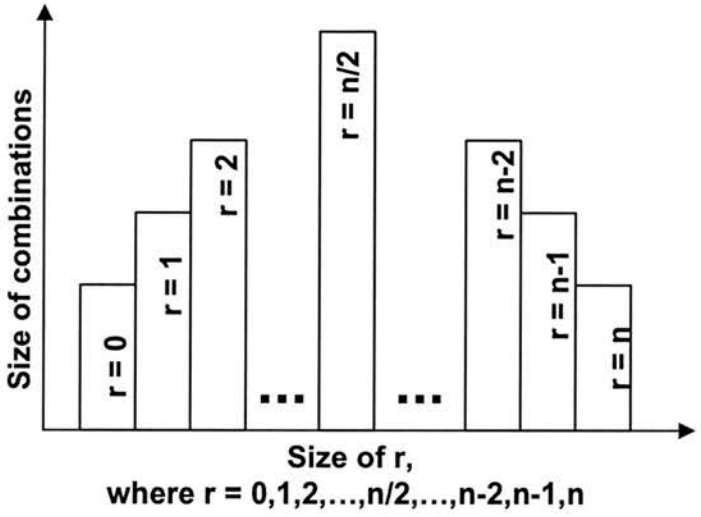


Figure 6.3: The size of combinations for different values of *r*

For the experiment, the portion of allowed domain values is set to either 25% or 50% of a given domain size. In the case where compatibility levels of local constraints are equivalent to a decimal-point value, it will be rounded-off to the nearest integer. These two compatibility levels of local constraints can be described as follows, using a universal domain consisting of four distinct values:

- At the 25% compatibility level of local constraints, only a single value is allowed to be assigned to each variable at a time. Given *r=1*, then C_4^1=4, which means that there exists four possible distinct set of values that could be used for value assignments.

- At the 50% compatibility level of local constraints, two values are allowed to be assigned to each variable at a time. Given $r=2$, then $C_4^2=6$, which means that there exists six possible distinct set of values that could be used for value assignments.

The 25% compatibility level of local constraints is considered a strict measure compared to the 50% compatibility level, as more combinations of values are able to be produced in the latter as compared in the former. In our experiment, the 0%, 75% and 100% compatibility levels are not tried since 0% gives agents empty domain and 100% has the effect of not having a local constraint. For a set of local constraints with a 75% compatibility level, the agents will agree on at least a single solution value for each variable of the MAP, as such, it is not possible to obtain an over-constrained state with this level of compatibility.

Based on these parameters, a total of 32 possible problem classes can be derived as described in the first column of tables 6.1 and 6.2. Each problem class is instantiated with an over-constrained MAP which consists of arbitrarily chosen CSPs, set to be inconsistent at the specified problem settings. For instance, a problem class of (25,80,4) indicates an over-constrained MAP which consists of a set of conflicting CSPs with a domain size of 4, where each CSP is prescribed to an individual agent. The conflict involves one variable and only a single solution value is allowable for each variable at a time.

During the execution of the protocol, each individual agent is required to provide a problem space, contained within a set of relaxed problems for solving the over-constrained MAP. Though the constraint relaxation strategies applied by each agent are private, for the purpose of this evaluation, we make a sensible assumption that each agent will generate a set of relaxed problems with a solution subset distance close to its original. There are various approaches we could adopt for simulating the generation of problem spaces by the agents. One basic approach is to randomly generate a set of relaxed problems given a required distance from the original CSPs. However, as emphasised in [Edvardson, '99; Belinfante et al., '05], a random generation approach lacks coverage and realism, i.e., most relaxed problems which are within the target distance are not generated since they unlikely happen at random. As such, we employ an

exhaustive, distance-guided approach for generating the problem spaces. In this approach, the set of relaxed problems contained in the agents' problem spaces are exhaustively generated for each distance level, beginning with the one having the closest distance to the originals. This will continue until the protocol reaches a completion state. The use of distance-guided technique is to ensure that the exhaustive means of problem generation is feasible, and to avoid the problem spaces from becoming explosively large and difficult to handle.

6.2.2 Distributed Constraint Relaxation Phase

The constraint relaxation process takes place among agents assuming the two roles prescribed in the protocol – *initiator* and *responder*. As described in chapter 3, there are two ways an interaction concerning multi-issue problem could be handled – the agents assuming these two roles can communicate all the variables together (i.e. batch processing) or one after the other (i.e. issue-by-issue processing). In the issue-by-issue processing, at any one time, only one variable is communicated between the agents. As such, testing against various external constraint compatibility percentages might not be possible in this construct. Therefore, to obtain a complete evaluation on how the protocol's fare given the prescribed settings, the over-constrained MAP is resolved using the batch processing.

6.3 Experimental Results

Tables 6.1 and 6.2 provide a summary on the results obtained from testing the protocol against different problem classes. Each problem class is described in the first column of the tables using parameters of (LC, EC, DS) where;

- LC – Compatibility level of local constraints (i.e. 25 and 50).
- EC – Compatibility level of external constraints (i.e. 20, 40, 60, and 80).
- DS – Domain size (i.e. 4, 5, 6, and 7).

The results are grouped based on the 25% and 50% compatibility levels of local constraints. These are described in tables 6.1 and 6.2 respectively. Given a different set of problem classes as shown in the first column, the tables provide the following in the subsequent columns:

- Second column – The number of relaxation cycles needed by the agents to reach a completion state of the protocol.
- Third column – The cardinality of the *relaxation_path* at the completion of the protocol execution. This cardinality indicates the number of relaxation cycles of the second column which are *reachable to solution*. A relaxation cycle is considered has achieved a *reachable to solution* state if the set of relaxed problems generated by the interacting agents in that particular cycle are MAP solvable and are obtained with the minimal solution subset distance of the fourth column.
- Fourth column – The minimal solution subset distance required for reaching a solvable MAP state. This is further classified into two sub-columns, namely G_{Total} and G_{Max}.

Problem class	Relaxation cycles for completing the protocol	Cardinality of *relaxation_path*	Solution subset distance	
			G_{Total}	G_{Max}
(25,80,4)	4	2	2	2
(25,60,4)	7	3	4	2
(25,40,4)	37	6	6	4
(25,20,4)	67	6	8	4
(25,80,5)	5	2	2	2
(25,60,5)	9	3	4	2
(25,40,5)	61	6	6	4
(25,20,5)	113	6	8	4
(25,80,6)	9	5	2	2
(25,60,6)	17	9	4	2
(25,40,6)	235	60	6	4
(25,20,6)	441	96	8	4
(25,80,7)	11	5	2	2
(25,60,7)	21	9	4	2
(25,40,7)	361	60	6	4
(25,20,7)	681	96	8	4

Table 6.1: Protocol's performance against over-constrained MAPs with 25% compatibility level of local constraints

Problem class	Relaxation cycles for completing the protocol	Cardinality of *relaxation_path*	Solution subset distance	
			G_{Total}	G_{Max}
(50,80,4)	5	5	2	2
(50,60,4)	9	9	4	2
(50,40,4)	64	60	6	4
(50,20,4)	117	96	8	4
(50,80,5)	7	5	2	2
(50,60,5)	13	9	4	2
(50,40,5)	136	60	6	4
(50,20,5)	253	96	8	4
(50,80,6)	10	10	2	2
(50,60,6)	19	19	4	2
(50,40,6)	298	270	6	4
(50,20,6)	554	486	8	4
(50,80,7)	13	10	2	2
(50,60,7)	25	19	4	2
(50,40,7)	523	270	6	4
(50,20,7)	985	486	8	4

Table 6.2: Protocol's performance against over-constrained MAPs with 50% compatibility level of local constraints

6.4 Analysis of Results

Though there already exists a number of works which integrate constraint satisfaction techniques (i.e. CSP/DCSP) within the multi-agent systems as described in the introductory chapter of this thesis, many of these works are either:

- Do not address the over-constrained problem; or
- They are based on the subjective-based coordination approach.

As far as we know of, the research reported in the thesis is the first to realise the distributed partial CSP technique for addressing an over-constrained problem using the objective-based coordination approach for multi-agent systems (i.e. LCC). Since there exists no standard benchmark to provide an empirical, vis-à-vis comparative study on the performance of our approach against other existing agent-based works for solving distributed, over-constrained problems, one of the feasible options is to empirically evaluate our approach using a set of generated problem instances with different hardness levels. Based on the results obtained in the evaluation, it can be generally concluded that our approach exhibits the common characteristics similar with the CSP/DCSP techniques used for addressing an over-constrained problem within the distributed problem solving environment – an increase in the hardness level of a problem requires more time for reaching a solution.

For a detailed discussion on the obtained results, we provide macro-level and micro-level analyses in sub-sections 6.4.1 and 6.4.2 respectively. In the macro-level analysis, the focus is on the overall view concerning the interactions between the different problem settings. In addition, we also view the results from a case-by-case perspective, focusing on different domain sizes for the micro-level analysis.

6.4.1 Macro-level Analysis

As illustrated in figures 6.4 and 6.5, there is an overall decrease in the number of relaxation cycles required by agents to fully complete their parts of the protocol as the compatibility level of external constraints is gradually increased from 20% to 80% in all classes of domain size. An over-constrained problem with a low compatibility level of external constraints (i.e. 20%) consists of more variables to be satisfied compared to those with a high compatibility level (i.e. 30% and above). As such, the former involves more relaxation cycles for reaching a completion state compared to the latter since a higher number of unsatisfied variables contained in an over-constrained MAP requires more relaxation interactions and computations to be performed by the agents before a solvable state is achieved.

The graphs illustrated in both figures (i.e. 6.4 and 6.5) describe an identical pattern, that is, an overall decrease of relaxation cycles in accordance to an increase in the compatibility level of external constraints. However, the class of over-constrained MAPs with a 50% compatibility level of local constraints described in figure 6.5 records more relaxation cycles. This is due to an increase in the number of relaxed problems that the agents are able to generate in their respective problem spaces as we expand the compatibility level of local constraints from 25% to 50%. As more relaxed problems are available in the problem spaces of the agents, it provides more options for obtaining a set of solvable MAPs with a minimal solution subset distance. A larger problem space means that more interactions and computations are required from the agents for finding all possible combinations of relaxed problems which are MAP solvable, and at the same time, produce a minimal solution subset distance at the global level.

Across the different domain sizes, over-constrained MAPs with a bigger domain size (e.g. 7) require more relaxation cycles for reaching a completion state as compared to those with a smaller domain size (e.g. 4). This is due to a higher number of relaxed problems available in the former, which increases the scale of interaction and computational processes across agents.

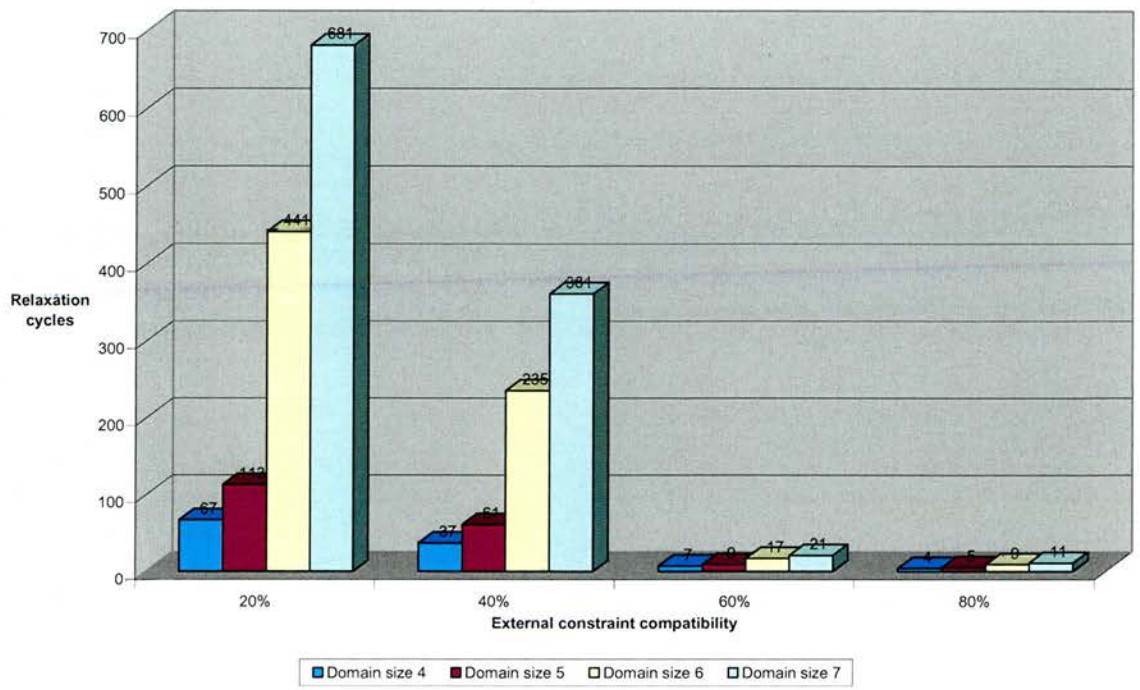


Figure 6.4: Relaxation cycles for over-constrained MAPs with 25% compatibility level of local constraints.

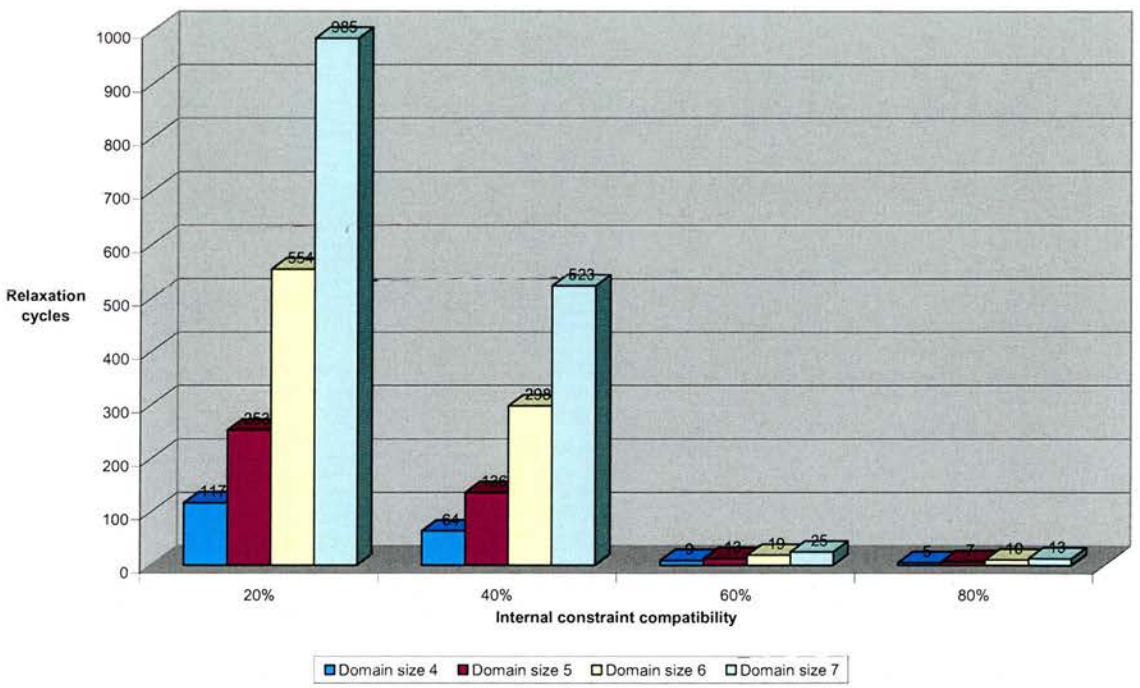


Figure 6.5: Relaxation cycles for over-constrained MAPs with 50% compatibility level of local constraints.

The overall general relationships between the three parameters and the number of relaxation cycles obtained from the protocol's execution are described by graphs (a), (b) and (c) in figure 6.6. Assuming other parameters remain fixed, the following conclusions are made with regard to each parameter:

- As described in figure 6.6(a), a higher number of relaxation cycles is required when the protocol is tested against a class of over-constrained MAPs with a low compatibility level of external constraints. The opposite is true when the test is conducted using over-constrained MAPs with a high compatibility level. At a low compatibility level, an increase in the number of external constraints agreeable by all agents allows a significant improvement in the number of relaxation cycles taken to reach a completion state. However, at a higher compatibility level, an increase in the number of external constraints agreeable by all agents only provides a small improvement. This can be attributed to the fact that the difficulty level of an over-constrained MAP is inversely related to the compatibility level of external constraints. The problem becomes significantly harder as the number of external constraints in conflict grows higher, and vice versa. The former requires a higher number of relaxation cycles for reaching a completion state as compared to the latter.

- As described in figure 6.6(b), the protocol requires a higher number of relaxation cycle for achieving a completion state at the 50% compatibility level of internal constraints, as compared to the 25% compatibility level. Given a distinct set of universal domains in which each contains a different set of domain values, a higher compatibility level means more combinations of domain values are available to be assigned to the over-constrained variables. Due to this, the agents' problem spaces, contained within all possible combinations of relaxed problem are increased in size. The exploration on these expanded problem spaces for finding a set of solvable MAPs with a minimal solution subset distance requires extensive interactions and computations. This is reflected by an increased number of relaxation cycles required for solving

over-constrained problems with a higher compatibility level of internal constraints.

- As described in figure 6.6(c), there is a direct relationship between the number of relaxation cycles required by the protocol for achieving a completion state and the distinct domain sizes of the over-constrained MAP. An increase in the domain size of the MAP is followed by an increase in the number of relaxation cycles required for solving the problem among agents. A bigger domain size means more possible combinations of domain values are available to be assigned to the variables of the MAP. This will increase the size of the agents' problem spaces, which will accordingly increase the scale of interaction and computational processes across agents. This means more relaxation cycles are required for the constraint relaxation process of an over-constrained MAP with a bigger sized domain values.

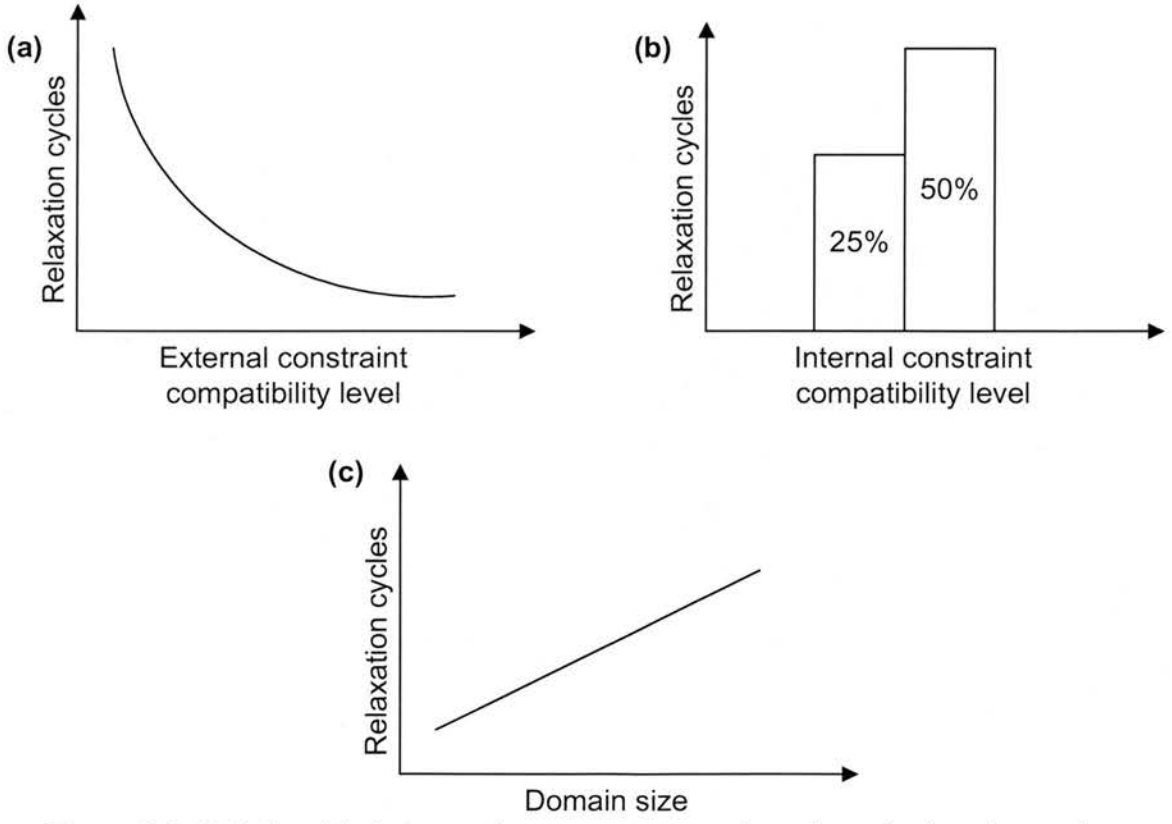


Figure 6.6: Relationship between the parameters and number of relaxation cycles obtained

6.4.2 Micro-level Analysis

In the micro-level analysis, the focus is on the following two aspects.

6.4.2.1 Margin of Difference for the Required Relaxation Cycles

In figures 6.7–6.10, we compare the total number of relaxation cycles obtained from executing the protocol against a set of over-constrained MAPs with different compatibility levels of local constraints. The comparison is made for each class of domain size where the 25% and 50% compatibility levels of local constraints are labelled as RC-25% and RC-50% respectively. As realised from the bar graphs in figures 6.7–6.10, the total number of relaxation cycles required by both problem classes is compounded inversely with the different levels of external constraints. Across each different compatibility level of external constraints (i.e. 20%–80%), RC-50% is always higher than RC-25%. However, there is a steep decrease in the margin of difference between the total relaxation cycles of these two levels as we gradually increase the compatibility level of external constraints from 20% to 80%. At the 20% and 40% external constraint compatibility levels, the differences are highly significant, however, at the 60% and 80% external constraint compatibility levels, the differences between RC-50% and RC-25% become negligible. This can be attributed to the following combined factors:

- RC-50% is regarded as a less stringent measure of the two, as such, it produces a more dense problem spaces compared to RC-25%. For both class of problems, as we gradually increase the compatibility level of external constraints from 20% to 80%, there is an acute decrease in the density of agents' problem spaces. In much harder problem settings (i.e. 20% and 40% external constraint compatibility levels), the problem space size of RC-50% is enormously large in comparison to RC-25%. However, as the problem settings become easier (i.e. 80% and 60% external constraint compatibility levels), the problem space size

of these two problem classes no longer has a significant difference. This will require fewer relaxation cycles to reach a termination state.

- The search for a solvable MAP state begins with relaxed problems nearest to the original CSPs. If no solution with a low global solution subset distance is found, explorations will proceed on relaxed problems further away from the originals. This means the task of deriving a solvable MAP state with a low solution subset distance value requires less computational and interactive efforts, compared to the one with a higher distance value. The correlation between the described cases (i.e. RC-25% and RC-50%) and the solution subset distance values (i.e. G_{Total} and G_{Max}) are shown in the fourth column of tables 6.1 and 6.2. The solution subset distance value for reaching a solvable MAP state decreases as the compatibility levels of external constraints increases across the different domain sizes. Furthermore, these distance values are consistent with the number of relaxation cycles obtained by both class of problems across the different compatibility levels of external constraints. An increase in the number of relaxation cycles needed for reaching a completion state is related with an increase in the solution subset distance values required for obtaining a solvable MAP given the different difficulty levels of over-constrained problems.

6.4.2.2 Ratio of Solvable Problems

In this experiment, we divide the relaxation cycles into two classes: cycles that are *reachable to solutions*, and cycles that are *unreachable to solutions* because either the relaxed problems obtained from the agents in a particular constraint relaxation cycle are non-solvable or the global solution subset distance derived from the relaxation process is higher than the existing one. The cardinality of *relaxation_path* is used to identify the relaxation cycles that are *reachable to solutions*, and this information is provided in the third column of tables 6.1 and 6.2. Based on this information, a pair of line graphs, labelled as Car–25% and Car–50%, are generated in figures 6.7–6.10 to respectively illustrate the protocol's performance against a class of over-constrained MAPs with 25% and 50% compatibility levels of local constraints. These graphs could be utilised to

determine the relationship and trend of change between 1) the number of relaxation cycles which are *reachable to solutions*; and 2) the total number of relaxation cycles required for reaching a protocol's completion state. This relationship, termed as the ratio of solvable problems, is achieved by comparing RC–25% with Car–25%, and RC–50% with Car–50%, for different compatibility levels of local constraints and across different classes of domain sizes. Based on the relationships shown by these bar and line graphs, the following observations can be made:

- At the 60% and 80% compatibility levels of external constraints, the ratio of solvable problems is significantly high.
- At the 20% and 40% compatibility levels of external constraints, the following are true:
    - There is a large increase in Car–50%, which consistently follows a steep increase in RC–50% for all problem classes. This indicates a relatively high ratio of solvable problems, which is particularly evident in problem classes of figures 6.7 and 6.9.
    - A large increase in RC–25% has no significant impact on the size of Car–25%. This is particularly evident in problem classes of figures 6.7, and 6.8. In figures 6.9 and 6.10, there is relatively a small increase in Car–25%, in response to a steep increase in RC–25%. These indicate that the ratio of solvable problems at the 25% compatibility level of internal constraints is relatively low.
    - Overall, the ratio of solvable problems at the 50% compatibility level of internal constraints is relatively high in comparison to the ratio of solvable problems at the 25% compatibility level for all problem classes.

In addition, as realised in the graphs and also in tables 6.1 and 6.2, a recurring pattern of identical Car–25% and Car–50% sizes are obtained for different problem classes that have the same number of allowable value assignments for each variable at the 25% and 50% compatibility levels of local constraints. This behaviour is expected given the

problem spaces generated by the interacting agents during the constraint relaxation process are simulated using an exhaustive, distance-guided approach. For instance, for the problem classes with a domain size of 4 and 5, at the 25% compatibility level of local constraints, only a single value is allowed to be assigned to each variable at a time. Having the similar number of allowable value assignments for each variable, the computation based on the solution subset distance heuristic on these problem classes within our current testing and evaluation construct will generate the *relaxation_path* of the same cardinality, given that the solution subset distance derived from the performed constraint relaxation is also the same.

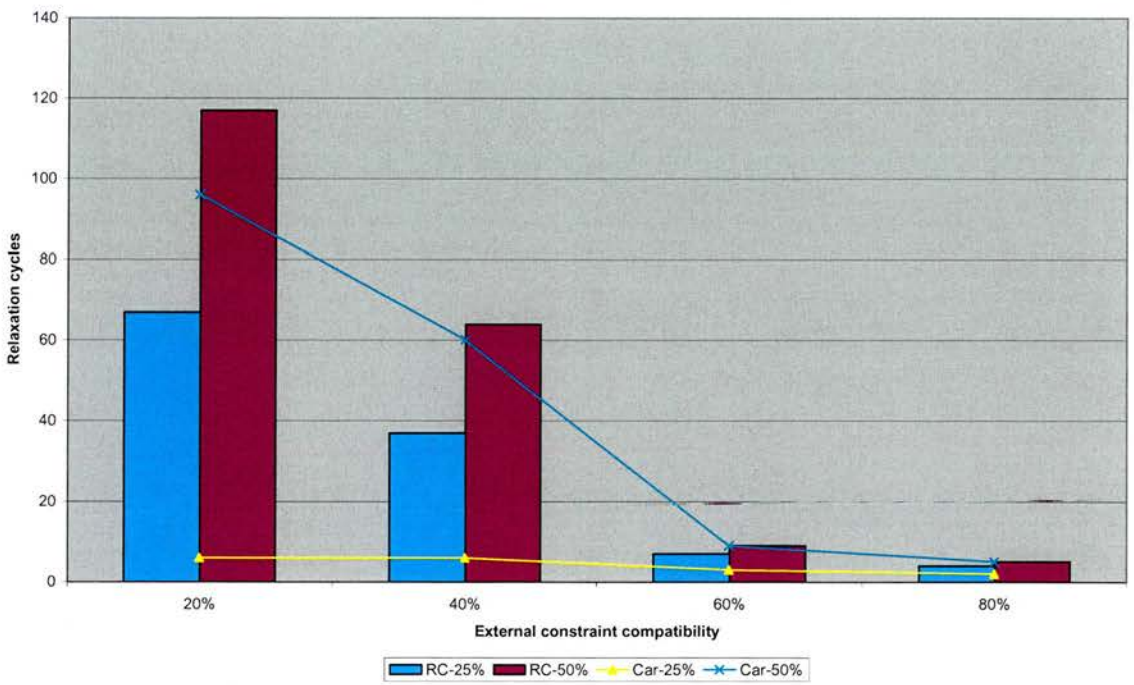


Figure 6.7: Comparison between 25% and 50% compatibility levels of local constraints– Over-constrained problem with a domain size of 4

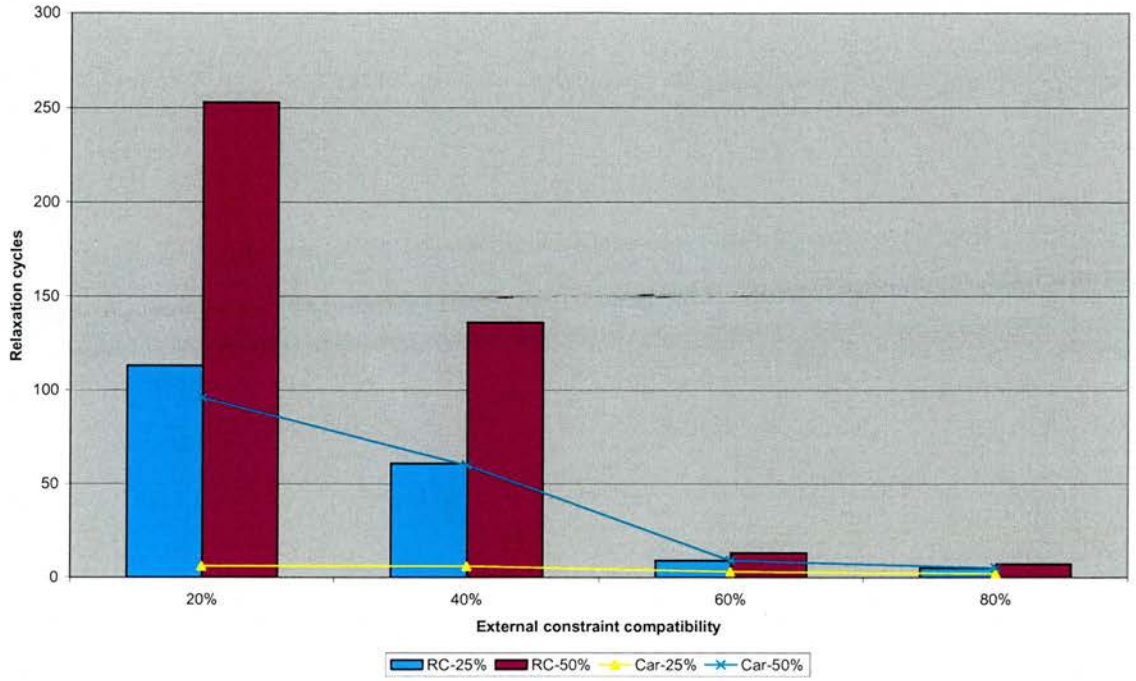


Figure 6.8: Comparison between 25% and 50% compatibility levels of local constraints– Over-constrained problem with a domain size of 5

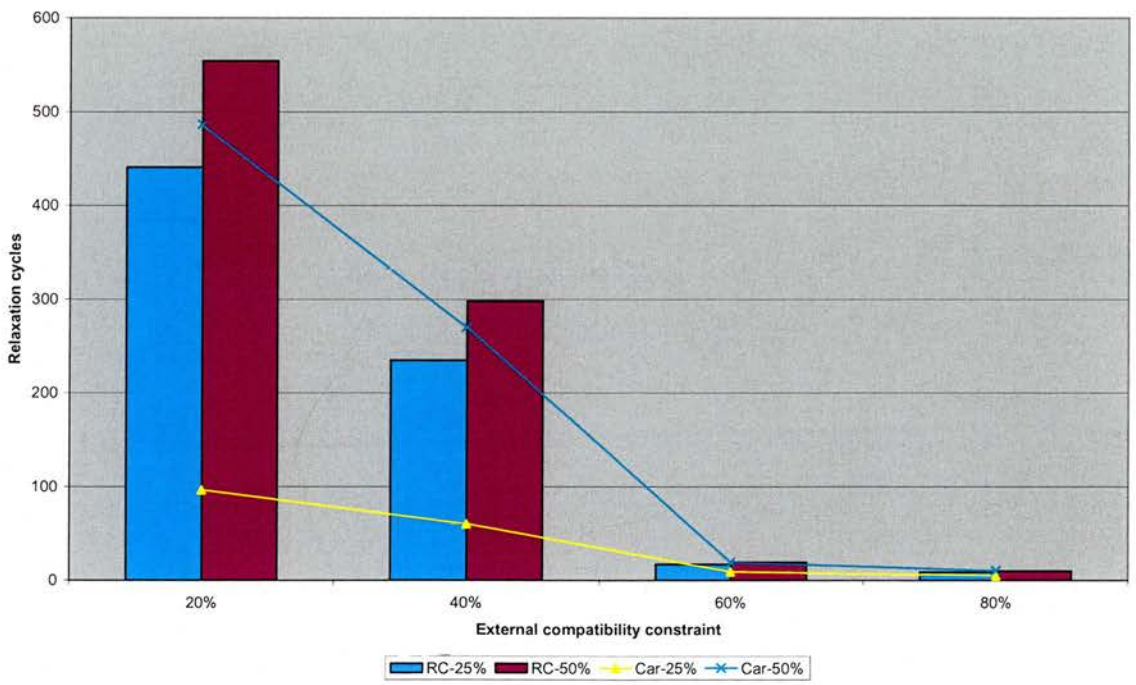


Figure 6.9: Comparison between 25% and 50% compatibility levels of local constraints– Over-constrained problem with a domain size of 6

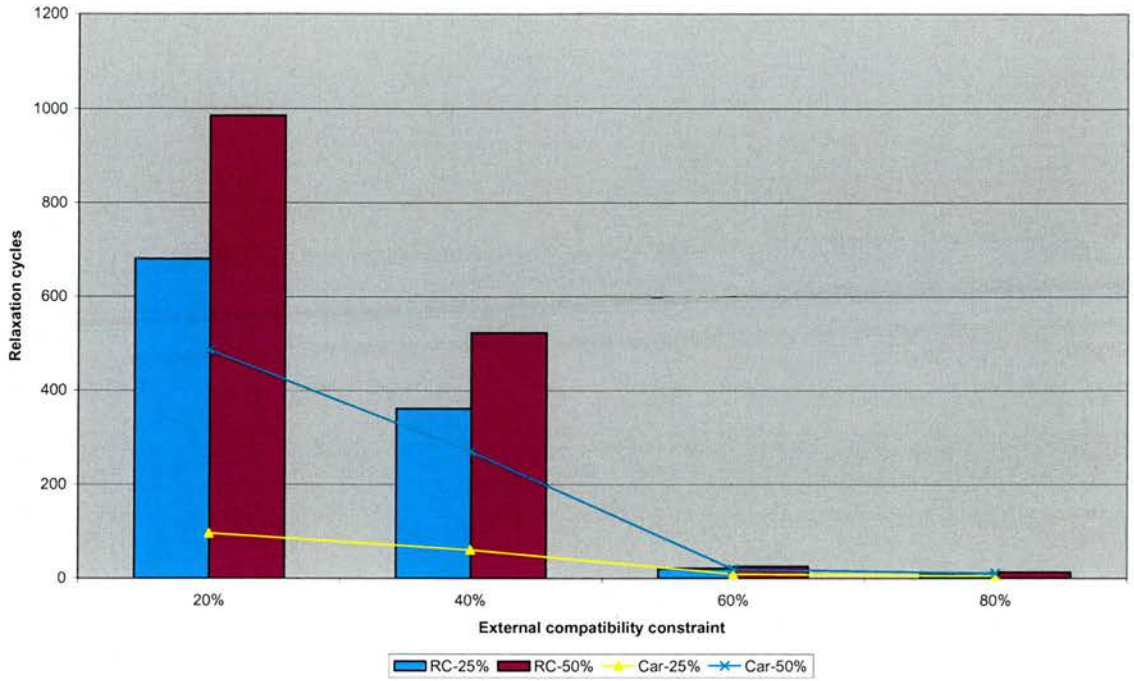


Figure 6.10: Comparison between 25% and 50% compatibility levels of local constraints –Over-constrained problem with a domain size of 7

6.5 Chapter Summary

In this chapter, we described the evaluation of the constraint relaxation protocol. For this purpose, in section 6.1, we provided a discussion on the adopted time-based measurement, and its advantages over other forms of measurement. In section 6.2, we presented the experimental test bed which consists of two phases; a) a problem generation phase of the over-constrained MAPs, and, b) a distributed constraint relaxation phase of the over-constrained MAPs via the protocol. We showed the results obtained from testing the protocol against different problem classes in section 6.3. Finally, in section 6.4, we provided macro-level and micro-level analyses on the obtained results.

Chapter 7

Conclusions and Future Works

This chapter provides the conclusions for the thesis and outlines areas which merit further investigations.

7.1 Conclusions

The thesis has shown that our primary goal is fulfilled; to address the brittleness of protocol-led agent interaction for solving distributed problems. As the distinct sub-problems of the individual agents are interdependent, the existence of an over-constrained state becomes the source of this brittleness. We have shown how a constraint relaxation approach can adopted, by realising the distributed partial CSP as an interaction protocol using the LCC. This allows heterogeneous agents, assumed to have the cognitive capability of relaxing their individual constraints, to take part in the interaction and coordination of distributed constraint relaxation process for obtaining a solvable state, if there exists one.

An important contribution of this thesis is not only realising the ideas of integrating distributed partial CSP with LCC, but also providing a practical and executable solution. The specification and execution of the protocol is achieved in a completely modular way, without needing to modify the LCC language or expansion engine. The only minimal requirement is to expand the existing LCC framework to include a constraint relaxation computational engine, which consists of axioms and inference procedures for performing constraint relaxation computations across agents. This additional component, which provides the necessary interface with a finite-domain constraint solver, is implemented in SICStus Prolog.

The time-based measurement is used to determine the protocol's performance, and this is achieved by analysing the number of cycles taken by the agents to complete their

respective parts in the protocol. For the experimental test bed, a set of over-constrained MAPs with different levels of hardness are generated to be tested against the protocol. The results have shown that a harder problem generally requires a higher number of cycles for reaching a completion state.

Not only does this thesis explore a flexible and novel approach of handling constraints within the interaction domain of heterogeneous and autonomous agents, but it is also grounded in a practical implementation. We have shown that our approach is not specifically engineered as part of the agents' internal reasoning mechanisms, and its deployment and execution does not rely on any centralised mechanism. In this way, the brittleness of agent interactions due to the conflicting constraints imposed by the individual agents can be addressed by the agents themselves without any third-party intervention. As such, any limitation associated with the third-party mediator approach could be safely avoided.

In addition, the research reported in the thesis has bridged the gap between established works from two separate research disciplines; the constraint satisfaction and distributed protocol for multi-agent systems. It has shown on how we could utilise the available technique in one research field to solve the problem of another. It benefits both disciplines in the following two general aspects.

1) For the constraint satisfaction research field, it makes the available techniques to address over-constrained problem relevant for the peer-to-peer agent environment.
2) For the multi-agent system research field, particularly the distributed agent protocol, it addresses the brittleness problem commonly faced by problem solving agents during their interactions for finding a solution.

Though this work is far from complete, it will pave a way for the integration of other available constraint satisfaction techniques based on fuzzy or probabilistic with the objective-based coordination approach of MAS (e.g. LCC) to allow agents to have flexible interactions in solving distributed, constrained problem.

7.2 Future Works

In this section, we describe possible improvements to the research work presented in the thesis. These improvements do not change the fundamental premise of the thesis – addressing the brittleness of protocol-led agent interaction for distributed problem solving, but rather provide additional means to expand upon the work completed and further areas of experimentation that are beyond the current work.

7.2.1 Employing Constraint Relaxation Strategies

The strategies employed by each agent during the constraint relaxation process is considered private. Given an over-constrained problem, the issue of the best computational approach or constraint relaxation strategy that an agent might employ for reaching a solvable state is still open, and its discussion extends beyond the scope of this thesis. Though the issue is not fully explored here, we fully acknowledge that one of the important experimentations is to evaluate the protocol against all possible constraint relaxation strategies that may be employed by the interacting agents during the constraint relaxation process. For this purpose, one of the possible future research work is to utilise the constraint relaxation strategies described in [Norlander et al., '03; Norlander, '04] for simulating the generation of problem spaces by the interacting agents. We could set the constraint relaxation strategies to be either uniform or varied across agents. A comprehensive and extensive system of experimentation concerning the relationship of the constraint relaxation strategies and the protocol performance could be established. Consequently, a general conclusion associating the protocol's performance and the specific constraint relaxation strategies employed by the agents could be drawn.

7.2.2 Utilising Different Distance Metrics

The degree of constraint relaxation performed by each agent for reaching a solvable state is computed based on the comparison made between its original, over-constrained CSP with the set of relaxed CSPs that allow this state to be achieved. Besides solution subset

distance, a number of other distance metrics that could be employed for this purpose include augmentation and Max-CSP [Bistarelli et al., '04]. These metrics differ in terms of the aspects used for the comparison.

As described in chapter 4, augmentation and Max-CSP fundamentally focus on the agents' manipulations on their local constraints for obtaining a solvable state. One of the major disadvantages with this approach is that it inadvertently reveals the agents' strategies as constraint details of the local problems need to be publicly and openly shared between the distributed agents. However, if the agents are working in the environment which regards privacy as non-critical and allows details concerning the local constraint specifications of each individual agent to be openly shared, then the augmentation and Max-CSP provide good alternatives to the currently utilised solution subset distance. Employing these constraint-based distance metrics opens up other interesting research issues. One particular concern is on the level of detail to be communicated among agents during the constraint relaxation process. Do we allow only a certain aspect of the constraints (e.g. number of violated constraints, degree of violation, etc.) to be carried and propagated with the protocol, or, can the individually defined constraint graph of each agent become public knowledge accessible by all agents at the protocol level? This depends on the level of trust [Ramchurn et al., '04] that the agents have towards their interacting partners. This becomes more complex when the level of trust across agents is conflicting with each other. How the different level of constraint details concerning the different levels of trust are modelled at the protocol level is an interesting research question to be explored.

7.2.3 Handling of Non-Crisp Constraints

Within the constraint community, a lot of effort has been devoted to extending the conventional notion of constraint, whose truth value is computed in a boolean (true/false) algebra, to be able to model features like fuzziness, uncertainty, optimisation, probability, and partial satisfaction. As described in [Rudova and Matyska, '99], various types of preferences, priorities, satisfaction degrees or weights were proposed to find solutions of over-constrained problems where some kind of relaxation have to be involved to get

feasible solutions. Two of these methods are possibilistic CSP [Schiex, '92] and fuzzy CSP [Dubois et al., '96], which support the representation of constraints as non-crisp relations. Possibilistic CSP assigns to each constraint some preference degree, which expresses necessity of its satisfaction. Fuzzy CSP considers each constraint as a relation, with different levels of preferences. Preference degrees in both methods are designed based on fuzzy sets, possibility theory and possibilistic logic. Assuming that the interacting agents are equipped with the described non-crisp form of formalisations at the local level, then it opens up a possibility for possibilistic CSP and fuzzy CSP techniques to be realised as LCC-based interaction protocols for addressing any over-constrained problem among agents. For this to work, it requires the interaction framework to be expanded to include mechanisms for accommodating the propagation of these forms of constraint formalisms across agents. This would enable us to support agent interaction for solving distributed problems of this nature.

7.2.4 Evaluation Based on Real-Life Applications

Within the constraint satisfaction research field, it is a common practice for constraint satisfaction techniques to be evaluated empirically using a set of generated problem instances with different difficulty levels. For evaluating our approach, we employed a similar method by developing a domain independent test bed that consists of features common to many distributed constraint solving problems regardless of domain. For future enhancement, the test bed could be expanded to include real-life applications, particularly in domains where exact solutions might be hard to find and partial solutions are tolerable. One of the possible options is to evaluate our approach using the distributed cooperative scheduling domain. Existing works within this domain that focus on the over-constrained problem include that of [Luo et al., '00], which proposed a fuzzy-based model and [Tsuruta and Shintani, '00], which employed a distributed values constraint satisfaction.

Appendix A

Prolog Code

A.1 interface.pl

This is the code that provides the interface between the loaded protocol, the expansion engine and the constraint relaxation computational engine. The code also specifies on how the message and protocol is loaded to/from the Linda. This code is adapted from the basic LCC framework.

```
/*************************************************************************
```
`institution/3` is used to load the constraint relaxation scene specified in the institution file. Given that protocol for the scene is specified in the institution file of *relaxation.inst*, the agent who is currently faced with an over-constrained state needs to assume the specified role of an initiator, with some specified Id (e.g. b1). The following is typed at the command line to enact the protocol:

institution(relaxation,initiator,b1).

Prot - The content of the LCC constraint relaxation protocol loaded from the institution file of *relaxation.inst* is divided into 3 lists with a syntactical form of def([],[],[]):

1st list - initially empty, later used to keep tracked of completed protocol state between agents, proc is closed (i.e. c(proc)) if the agents' part as specified in the protocol is complete
2nd list - loaded protocol
3rd list - loaded common knowledge

`react/1` is a predicate used to achieve the following goals:
1) Retrieve the intended message and protocol for the agent of given Id from the Linda
2) Display the retrieved message on the screen
3) Call to **`postit/3`**

Thus, to retrieve and process message intended for the responder agent (i.e. of Id s1) the following is typed at the command line:

react(s1).

`postit/3` is a predicate used to achieve the following goals:
1) Expand the received message and protocol using the rewrite rules
2) Display the message obtained from (1) on the screen, to response to the message received from the other interacting agents
3) Post the intended message for the other agents in the utilised Linda.

```
*************************************************************************/
```

```
%*****************************************************************************

institution(I, Role, Id) :-
    load_institution(I, Prot),!,
    postit(a(Role,Id), [], Prot).

react(Id) :-
    retrieve_message(_, Id, Dialogue),
    Dialogue = protocol(m(Af,M => At),Prot),!,
    postit(At, [m(At,M <= Af)], Prot).

postit(Role, IMessages, Prot) :-
    expansion(Role, IMessages, [], Prot, RMessages, Messages, EProt),
    RMessages = [],!,
    send_protocol_messages(Messages, EProt).

%*****************************************************************************

agent_id_from_role(a(_,Id), Id).

send_protocol_messages([m(Af,M => At)], Prot) :-
    agent_id_from_role(Af, From), nonvar(From),
    agent_id_from_role(At, To), nonvar(To),
    send_message(From, To, protocol(m(Af,M => At),Prot)),!,
    react(To).

send_protocol_messages([],_).

send_message(From, To, Message) :-
    find_server(Server, PID),
    add_message(Server, PID, From, To, Message),
    Message = protocol(m(_,M => _),_),
    write('Outgoing msg: '), portray_clause(M), nl,!.

retrieve_message(From, To, Message) :-
    find_server(Server, PID),
    read_message(Server, PID, From, To, Message),
    Message = protocol(m(_,M => _),_),
    write('Incoming msg: '), portray_clause(M), nl,!.

%*****************************************************************************

load_institution(Institution, InstDef) :-
    concat(Institution,'.inst',File),
    see(File),
    read_institution(InstDef),
    seen.
```

```
%*****************************************************************************

read_institution(InstDef) :-
    read_institution1(def([],[],[]), InstDef).

read_institution1(InstDef, FinalInstDef) :-
    read(Clause),
    \+ Clause = end_of_file, !,
    add_to_institution_def(Clause, InstDef, NewInstDef),
    read_institution1(NewInstDef, FinalInstDef).
read_institution1(InstDef, InstDef).

add_to_institution_def((Head ::= Body),
                       def(I,D,K),
                       def(I,D1,K)) :-
    append(D, [(Head ::= Body)], D1).
add_to_institution_def(known(Agent,Clause),
                       def(I,D,K),
                       def(I,D,K1)) :-
    append(K, [known(Agent,Clause)], K1).

%*****************************************************************************
```

A.2 expansion_engine.pl

This is the code in which the expansion engine is specified. Built-in predicates are imported from the SICStus Prolog library of terms, lists and finite-domain constraint solver whenever necessary. The code is fundamentally adapted from the basic framework of LCC.

```
%*************************************************************************

:- op(900, xfx, '::='),
   op(900, xfx, '::'),
   op(900, xfx, '>>'),
   op(800, xfx, '=>'),
   op(800, xfx, '<='),
   op(830, xfx, '<--'),
   op(820, xfy, and),
   op(850, xfy, par),
   op(850, xfy, then),
   op(850, xfy, or).

%*************************************************************************
%Starting the expansion process.
%*************************************************************************


expansion(Agent, Ms, Os, P, FinalMs, FinalOs, FinalP, FDRange) :-
    expansion_step(Agent, Ms, Os, P, NewMs, NewOs, NewP, FDRange),
    expansion(Agent, NewMs, NewOs ,NewP, FinalMs, FinalOs, FinalP,
              FDRange).

expansion(Agent, Ms, Os, P, Ms, Os, P,FDRange) :-
    \+ expansion_step(Agent, Ms, Os, P, _, _, _, FDRange).

%*************************************************************************
%Selecting the protocol clause to expand and saving it after expansion.
%*************************************************************************

expansion_step(a(Role,Id), Ms, Os, P, NewMs, NewOs, NewP, FDRange) :-
    protocol_select(agent, P, (a(ARole,Id) ::= Def), P1),
    expand_protocol((a(ARole,Id) ::= Def), Role, Id, Ms, Os, P1, NewA,
                    NewMs, NewOs, P2,  FDRange),
    protocol_add(agent, P2, NewA, NewP).

expansion_step(a(Role,Id), Ms, Os, P, NewMs, NewOs, NewP, FDRange) :-
    \+ protocol_select(agent, P, (a(_,Id) ::= _), _),
    protocol_member(dialogue, P, Clause),
    Clause = (a(Role,Id) ::= Def),
    expand_protocol((a(Role,Id) ::= Def), Role, Id, Ms, Os, P,
                    NewA, NewMs, NewOs, P2, FDRange),
    protocol_add(agent, P2, NewA, NewP).
```

```
%*************************************************************************
%The rewrite rules
%*************************************************************************

expand_protocol(Var, _, _, Ms, Os, P, Var, Ms, Os, P, FDRange) :-
    var(Var), !.

expand_protocol(Role ::= Def, _, Id, Ms, Os, P, Role ::= E, Mf, Of, Pf,
                FDRange) :-
    expand_protocol(Def, Role, Id, Ms, Os, P, E, Mf, Of, Pf, FDRange).

expand_protocol(A or _, Role, Id, Ms, Os, P, E, Mf, Of, Pf, FDRange) :-
    expand_protocol(A, Role, Id, Ms, Os, P, E, Mf, Of, Pf, FDRange).

expand_protocol(_ or B, Role, Id, Ms, Os, P, E, Mf, Of, Pf, FDRange) :-
    expand_protocol(B, Role, Id, Ms, Os, P, E, Mf, Of, Pf, FDRange).

expand_protocol(A then B, Role, Id, Ms, Os, P, EA then B, Mf, Of, Pf,
                FDRange) :-
    expand_protocol(A, Role, Id, Ms, Os, P, EA, Mf, Of, Pf, FDRange).

expand_protocol(A then B, Role, Id, Ms, Os, P, A then EB, Mf, Of, Pf,
                FDRange) :-
    closed(A),
    expand_protocol(B, Role, Id, Ms, Os, P, EB, Mf, Of, Pf, FDRange).

expand_protocol(C <-- M <= A, Role, Id, Ms, Os, P, c(M <= A), Mf, Os,
                Pf, FDRange) :-
    select(m(Role,M <= A), Ms, Mf),
    satisfied(Id, P, C, Pf).

expand_protocol(M => A <-- C, Role, Id, Ms, Os, P, c(M => A), Ms,
                [m(Role,M => A) | Os], Pf, C):-
     satisfied(Id, P, C, Pf).

expand_protocol(M <= A, Role, _, Ms, Os, P, c(M <= A), Mf, Os, P,
                FDRange) :-
    select(m(Role,M <= A), Ms, Mf).

expand_protocol(M => A, Role, _, Ms, Os, P, c(M => A), Ms,
                [m(Role,M => A) | Os], P, FDRange).

expand_protocol(Role <-- C, _, Id, Ms, Os, P, Role ::= Def, Ms, Os, Pf,
                FDRange) :-
    Role = a(_,_),
    satisfied(Id, P, C, Pf),
    protocol_member(dialogue, P, (Role ::= Def)).

expand_protocol(Role, _, _, Ms, Os, P, Role ::= Def, Ms, Os, P,
                FDRange) :-
    Role = a(_,_),
    protocol_member(dialogue, P, (Role ::= Def)).

expand_protocol(null <-- C, _, Id, Ms, Os, P, c(null), Ms, Os, Pf,
                FDRange) :-
    satisfied(Id, P, C, Pf).
```

```
expand_protocol(null, _, _, Ms, Os, P, c(null), Ms, Os, P, FDRange).

%*************************************************************************
%Testing for closed or failed clauses
%*************************************************************************

closed(Var) :-
    var(Var), !, fail.
closed(c(_)).
closed(A or _) :-
    closed(A).
closed(_ or B) :-
    closed(B).
closed(A then B) :-
    closed(A),
    closed(B).
closed(A par B) :-
    closed(A),
    closed(B).
closed(_ ::= Def) :-
    closed(Def).

%*************************************************************************
%Testing for satisfied constraints predicates
%*************************************************************************

satisfied(Id, P, A and B, Pf) :- !,
    satisfied(Id,  P, A, Pn),
    satisfied(Id,  Pn, B, Pf).

satisfied(Id, P, X, Pf) :-
    meta_pred(Id, X, P, Pf, Call), !,
    Call.

satisfied(Id, P, absorb_protocol(P1,Role,Clause), Pf) :-
    disjoint_protocols(P, P1),
    protocol_member(dialogue, P1, Clause),
    Clause = (a(Role,Id) ::= _),
    merge_protocols(P, P1, Pf).

satisfied(Id, P, X, P) :-
    \+ meta_pred(Id, X, P, _, _),
    call_direct(X),
    X.

satisfied(Id,  P, X, P) :-
    protocol_member(common_knowledge, P, known(Id, X)).

satisfied(Id, P, X, Pf) :-
    protocol_member(common_knowledge, P, known(Id, X <-- C)),
    satisfied(Id, P, C, Pf).
```

```
call_direct(X) :-
    (predicate_property(X, built_in) ;
     predicate_property(X, interpreted) ;
     predicate_property(X, imported_from(_))), !.

meta_pred(Id, not(X), P, P, \+ satisfied(Id,P,X,_)).
meta_pred(Id, retract(X), P, Pf,
          protocol_remove(common_knowledge,P,known(Id,X),Pf)).
meta_pred(Id, assert(X), P, Pf,
          protocol_add(common_knowledge,P,known(Id,X),Pf)).
```

%Meta-predicate concerning the specification of computational process for constraint relaxation
%to be taken by agents. Detailed specifications of the called predicates are provided in the
%constraint_handling.pl

```
meta_pred(_,assign(Suff,Prob,NSuff),P,P,assign(Suff,Prob,NSuff)).
meta_pred(_,add(Existing,Selected,HList),P,P,
        add(Existing,Selected,HList)).
meta_pred(_,g_solvable(RPath),P,P,g_solvable(RPath)).
meta_pred(_,g_distance(RPath,NEPath),P,P,
        g_distance(RPath,NEPath)).
meta_pred(_,find_solvable(PS,Suff,Sl),P,P,
        find_solvable(PS,Suff,Sl)).
meta_pred(_,distance_computation(Sl,O,Dl),P,P,
        distance_computation(Sl,O,Dl)).
meta_pred(_,select_minimal(DisTPS,Minimal,Agent),P,P,
        select_minimal(DisTPS,Minimal,Agent)).
meta_pred(_,response_composition(Minimal,Suff,Is,D),P,P,
        response_composition(Minimal,Suff,Is,D)).
meta_pred(_,revised_suff(NSuff,Respond,USuff),P,P,
        revised_suff(NSuff,Respond,USuff)).
meta_pred(_,invalid_spec_removal(PS,NHList,TPS),P,P,
        invalid_spec_removal(PS,NHList,TPS)).
meta_pred(_,path_computation(RPath,Minimal,TSuff,NEPath,TEPath),
        P,P,path_computation(RPath,Minimal,TSuff,NEPath,TEPath)).

%*************************************************************************
%Managing the protocol predicates
%*************************************************************************

closed_dialogue(Role, Prot) :-
    \+ ( protocol_member(agent, Prot, a(Role,_) ::= Def),
         \+ closed(Def) ).

disjoint_protocols(P1, P2) :-
    \+ ( protocol_member(dialogue, P1, a(Role1,_) ::= _),
         protocol_member(dialogue, P2, a(Role2,_) ::= _),
         functor(Role1, F, A),
         functor(Role2, F, A) ).

merge_protocols(def(A1,D1,K1), def(A2,D2,K2), def(A3,D3,K3)) :-
    append(A1, A2, A3),
    append(D1, D2, D3),
    append(K1, K2, K3).
```

```
protocol_component(agents, def(Clauses, _, _), Clauses).
protocol_component(dialogue, def(_, Clauses, _), Clauses).
protocol_component(common_knowledge, def(_, _, Clauses), Clauses).

protocol_member(agent, def(Clauses,_,_), Clause) :-
    member(Clause, Clauses).
protocol_member(dialogue, def(_,Clauses,_), ClauseCopy) :-
    member(Clause, Clauses),
    copy_term(Clause, ClauseCopy).
protocol_member(common_knowledge, def(_,_,Clauses), ClauseCopy) :-
    member(Clause, Clauses),
    copy_term(Clause, ClauseCopy).

protocol_select(agent, def(Clauses,A,B), Clause, def(R,A,B)) :-
    select(Clause, Clauses, R).
protocol_select(dialogue, def(A,Clauses,B), ClauseCopy, def(A,R,B)) :-
    select(Clause, Clauses, R),
    copy_term(Clause, ClauseCopy).
protocol_select(common_knowledge, def(A,B,Clauses), ClauseCopy,
                def(A,B,R)) :-
    select(Clause, Clauses, R),
    copy_term(Clause, ClauseCopy).

protocol_remove(agent, def(Clauses,A,B), Clause, def(R,A,B)) :-
    select(Clause, Clauses, R).
protocol_remove(dialogue, def(A,Clauses,B), Clause, def(A,R,B)) :-
    select(Clause, Clauses, R).
protocol_remove(common_knowledge, def(A,B,Clauses), Clause, def(A,B,R))
                :-
    select(Clause, Clauses, R).

protocol_add(agent, def(Clauses,A,B), X, def([X|Clauses],A,B)).
protocol_add(dialogue, def(A,Clauses,B), X, def(A,[X|Clauses],B)).
protocol_add(common_knowledge, def(A,B,Clauses), X,
            def(A,B,[X|Clauses])).
```

A.3 constraint_handling.pl

Prolog code for the detailed specification of constraint handling functionality. A number of predicates are imported from the pre-defined libraries of finite-domain constraint solver and list operations that come with SICStus Prolog.

```
%*************************************************************************
% Assignment & revision of sufficient bound after each relaxation cycle

assign(Suff,csp(_,SolSet),NSuff):-
      suff_assign(Suff,SolSet,NSuff),!.

assign(Suff,csp(_,SolSet,_),NSuff):-
      suff_assign(Suff,SolSet,NSuff),!.

suff_assign([],_,[]).
suff_assign([fd_term(Att,_)|T],SolSet,NSuff):-
      member(Sol,SolSet),
      Sol=fd_term(SAtt,SS),
      similar_term(Att,SAtt),!,
      NSuff=[fd_term(Att,SS)|R],
      select(Sol,SolSet,Next),
      suff_assign(T,Next,R).

similar_term(Att,PAtt):-
      Att=PAtt.

%*************************************************************************
%Updates on history list of an already selected problem. Selected problem in form of
%csp(Problem, SolutionSet)

add(Existing,Selected,HList):-
      append(Existing,Selected,HList).

%*************************************************************************
%To determine whether constraint relaxation path obtained so far is solvable or not - A path is
%solvable if the respond received by the initiator does not consist of any nil values, and the
%solutions for the sufficient bound variables produced by the agents intersect with each %other
(non-intersection is indicated by nil in-receipt response)

g_solvable(RPath):-
      \+ member(r(nil,nil),RPath),!.

%*************************************************************************
%To determine the global distance of the constraint relaxation path obtained so far

g_distance(RPath,NEPath):-
      distance_list(RPath,DL),!,
      sum_list(DL,Sum),
      max_list(DL,Max),
      NEPath=gdis(RPath,Sum,Max).
```

```
distance_list([],[]).
distance_list([r(_,D)|T],DL):-
      DL=[D|R],
      distance_list(T,R).

%*************************************************************************
%To determine whether the relaxed problem produced by the agents is solvable or not - within
%the necessary & sufficient bound

find_solvable([],_,[]).
find_solvable([CSP|Rest],Suff,Sl):-
      (
        (solvable(CSP,Suff),Sl=[CSP|R]);
        (\+ solvable(CSP,Suff),Sl=R)
      ),
      find_solvable(Rest,Suff,R).

solvable(csp(_,Sol,_),Suff):-
      is_intersect(Suff,Sol).

%*************************************************************************
%Checking on whether each similar variables of the relaxed problem & the sufficient %bound
intersect with each other

is_intersect([],_).
is_intersect([fd_term(Att,S)|R],PFd_Set):-
      member(Fd_Term,PFd_Set),
      Fd_Term=fd_term(PAtt,PS),
      similar_term(Att,PAtt),!,
      fdset_intersect(S,PS),
      select(Fd_Term, PFd_Set, PFd_SetRest),
      is_intersect(R,PFd_SetRest).

%*************************************************************************
%To select the most minimal relaxed problem in terms of solution subset distance from the
%original

select_minimal(DisTPS,Minimal,Agent):-
      acc_distance(DisTPS,DL),!,
      (
        (\+ DL=[],
            min_list(DL,Min),
            nth(Pos,DL,Min),!,
            nth(Pos,DisTPS,Minimal));

        ( DL=[], Minimal=[])
      ).

acc_distance([],[]).
acc_distance([csp(_,_,D)|Rest],Dl):-
      Dl=[D|Next],
      acc_distance(Rest,Next).
```

```
%*************************************************************************
%Composition of the constraint relaxation message

response_composition(Minimal,Suff,Is,D):-
       (
         Minimal=csp(_,Sol,D),
         suff_int_sols(Suff,Sol,Is));

       (Minimal=[],
        Is=nil,D=nil).

suff_int_sols([],_,[]).
suff_int_sols([fd_term(Att,S)|R],Sol,Is):-
       member(Fd_Term,Sol),
       Fd_Term=fd_term(PAtt,PS),
       similar_term(Att,PAtt),!,
       fdset_intersection(S,PS,Int),
       Is=[fd_term(Att,Int)|Next],
       select(Fd_Term,Sol,NSol),
       suff_int_sols(R,NSol,Next).

%*************************************************************************
%Revision of the necessary bound upon receipt of response from neighbouring agents

revised_suff(NSuff,Respond,USuff):-
       Respond=r(IntSol,_),
       (
         (IntSol=nil,
          USuff=NSuff);

        (\+ IntSol=nil,
            USuff=IntSol)
       ).

%*************************************************************************
%Pruning of the problem space generated through systematic elimination of relaxed problem that
%has already been visited or not comply with the necessary bound

invalid_spec_removal([],_,[]).
invalid_spec_removal([CSP|R],NHList,TPS):-
       (
         ( \+ member(CSP,NHList),
                    TPS=[CSP|Next]);
         ( TPS=Next)),
       invalid_spec_removal(R,NHList,Next).
```

```
%*************************************************************************
%Elimination of any relaxed problem from an already pruned problem  space that has a computed
%solution subset distance of more than the distance of an the currently selected relaxation path

invalid_dist_removal([],_,[]).
invalid_dist_removal([csp(_,S,D)|R],NEPath,FPS):-
      (
        ( locally_better_or_equal(D,NEPath),
          FPS=[csp(_,S,D)|Next]);

          FPS=Next),
      invalid_dist_removal(R,NEPath,Next).

locally_better_or_equal(D,gdis(_,T,G)):-
      (D<T);(D=:=G,D=<G).

%*************************************************************************
%Perform computation on the obtained relaxation path to determine whether the newly acquired
%path is better than the current path in store

path_computation(RPath,Minimal,NSuff,ESuff,TSuff,NEPath,TEPath):-
      response_composition(Minimal,ESuff,Is,D),
      append([r(Is,D)],RPath,UPath),
      g_distance(UPath,NRPath),
      (
        (\+ var(NEPath),
            select_path(NSuff,ESuff,TSuff,NRPath,NEPath,TEPath));

        ( var(NEPath),
          TEPath=NRPath,
          TSuff=ESuff)
      ).

select_path(NSuff,ESuff,TSuff,NRPath,NEPath,TEPath):-
      (
        ( better(NRPath,NEPath),
          TEPath=NRPath,
          TSuff=ESuff,
          print_progress(TEPath,TSuff));

        (TEPath=NEPath,
         TSuff=NSuff,
         print_progress)).

better(gdis(_,Tn,Gn),gdis(_,T,G)):-
      (Tn < T);
      (Tn=:=T,Gn=<G).
```

Bibliography

[Aldea et al., '01] Aldea, A., Lopez, B., Moreno, A., Riano, D. and Valls, A. A multi-agent system for organ transplant coordination. In Proceedings of the *VIII European Conference on AI in Medicine*, Carcais, Portugal, **LNCS (2101)**, 413-416, Springer Verlag, 2001.

[Ando et al., '03] Ando, M., Nato, M. and Toyoshima, H. Empirical evaluation of distributed maximal constraint satisfaction method. In Proceedings of the *IEEE International Conference on Systems, Man and Cybernetics*, Washington DC, USA, 4672-4677, 2003.

[Austin, '62] Austin, J. L. *How to do things with words*. Oxford University Press, Oxford, 1962.

[Belinfante et al., '05] Belinfante, A., Frantzen, L. and Schallhart, C. Tools for test case generation. *Model-based testing of reactive systems*. M. Broy, B. Jonsson et al, Eds., **LNCS (3472)**: 391-438, Springer Verlag, 2005.

[Bistarelli et al., '04] Bistarelli, S., Freuder, E. C. and O'Sullivan, B. Encoding partial constraint satisfaction in the semiring-based framework for soft constraints. In Proceedings of the *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'04)*, Boca Raton, FL, USA, 2004.

[Bocchi and Ciancarini, '03] Bocchi, L. and Ciancarini, P. A perspective on multiagent coordination models. *Communications in Multiagent System*. M.-P. Huget, Ed., **LNAI (2650)**: 146-163, Springer Verlag, 2003.

[Bond and Gasser, '88] Bond, A. and Gasser, L. *Readings in distributed artificial intelligence*. Morgan Kauffman, San Mateo, California, 1988.

[Borning et al., '92] Borning, A., Freeman-Benson, B. and Wilson, M. Constraint hierarchies. *Lisp and Symbolic Computation*. **5**: 223-270, 1992.

[Bratman, '87] Bratman, M. E. *Intentions, plans and practical reason*. Harvard University Press, Cambridge, MA, 1987.

[Brito et al., '04] Brito, I., Herrero, F. and Meseguer, P. On the evaluation of DisCSP algorithms. In Proceedings of the *Workshop on Distributed Constraint Reasoning, at the 10th International Conference on Principles and Practice of Constraint Programming (CP'04)*, Toronto, Canada, 2004.

[Cabri et al., '04] Cabri, G., Ferrari, L. and Zambonelli, F. Role-based approaches for engineering interactions in large-scale multi-agent systems. *Software Engineering*

for Multi-Agent Systems III. P. d. Lucena, Garcia et al, Eds., **LNCS (2940)**: 243-263, Springer Verlag, 2004.

[Cabri et al., '02] Cabri, G., Leonardi, L. and Zambonelli, F. Modelling role-based interactions for agents. In Proceedings of the *Workshop on Agent-Oriented Methodologies, at OOPSLA'02*, Seattle, USA, 2002.

[Calisti, '02] Calisti, M. Abstracting communication in distributed agent-based systems. In Proceedings of the *Concrete Communication Abstractions of the Next Distributed Object Systems Workshop, at the 16th European Conference on Object-Oriented Programming (ECOOP'02)*, Malaga, Spain, 2002.

[Calisti and Neagu, '04] Calisti, M. and Neagu, N. Constraint satisfaction techniques and software agents. In Proceedings of the *AIIA 2004 Workshop on Agents and Constraints*, Perugia, Italy, 2004.

[Carlsson et al., '97] Carlsson, M., Ottoson, G. and Carlson, B. An open-ended finite domain constraint solver. In Proceedings of the *9th International Symposium of Programming Languages, Implementations, Logics, and Programs (PLILP'97)*, Southampton, UK, H. Glaser, P. Hartel et al, Eds., **LNCS (1292)**, Springer Verlag, 1997.

[Carrieno and Gelernter, '89] Carrieno, N. and Gelernter, D. Linda in context. *Communications of the ACM.* **32**(4): 444-458, 1989.

[Chaib-Draa and Dignum, '02] Chaib-Draa, B. and Dignum, F. Trends in agent communication language. *Computational Intelligence.* **18**(2): 89-101, 2002.

[Chalmers, '04] Chalmers, S. S. *Agents and constraint logic.* PhD Thesis, Department of Computing, University of Aberdeen, 2004.

[Chen and Sadaoui, '03] Chen, B. and Sadaoui, S. *A generic formal framework for multi-agent interaction protocols.* Technical Report TR 2003-05, University of Regina, Canada, 2003.

[Chnadrasekaran et al., '99] Chnadrasekaran, B., Josephson, R. and Richard, V. What are ontologies and why do we need them? *IEEE Intelligent Systems.* **14**(1): 20-26, 1999.

[Davin and Modi, '05] Davin, J. and Modi, P. J. Impact of problem centralization in distributed constraint optimization algorithms. In Proceedings of the *Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, Netherlands, 2005.

[Davin and Modi, '06] Davin, J. and Modi, P. J. Hierarchical variable ordering for multiagent agreement problem. In Proceedings of the *Seventh International*

Workshop on Distributed Constraint Reasoning, at AAMAS'06, Hakodate, Japan, 2006.

[Decker et al., '88] Decker, K. S., Durfee, E. H. and Lesser, V. *Evaluating research in cooperative distributed problem solving*. Computer Science Technical Report 88-99, University of Massachusetts at Amherst, 1988.

[deSilva, '02] deSilva, L. P. *Extending agents by transmitting protocols in open systems*. Honours Thesis, RMIT University, Melbourne, 2002.

[Doran et al., '97] Doran, J. E., Franklin, S., Jennings, N. R. and Norman, T. J. On cooperation in multi-agent systems. *The Knowledge Engineering Review*. **12**(3): 309-314, 1997.

[Dubois et al., '96] Dubois, D., Fargier, H. and Prade, H. Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty. *Applied Intelligence*. **6**: 287-309, 1996.

[Edvardson, '99] Edvardson, J. A survey on automatic test data generation. In Proceedings of the *Second Conference on Computer Science and Engineering*, Linkoping, Sweden, 21-28, 1999.

[Edwards, '04] Edwards, A. W. F. *Cogwheels of the mind: The story of Venn diagram*. John Hopkins University Press, Baltimore & London, 2004.

[Esteva et al., '02] Esteva, M., Padget, J. and Sierra, C. Formalizing a language for institutions and norms. *Intelligent Agents VIII, LNAI*. **2333**: 348-366, 2002.

[Esteva et al., '00] Esteva, M., Rodrigez-Aguilar, J. A., Arcos, J. L., Sierra, C. and Garcia, P. Institutionalising open multi-agent systems. In Proceedings of the *Fourth International Conference on Multiagent Systems (ICMAS'2000)*, Boston, USA, 381-383, 2000.

[Esteva et al., '01] Esteva, M., Rodriguez, J. A., Sierra, C., Garcia, P. and Arcos, J. L. On the formal specifications of electronic institutions. *Agent Mediated Electronic Commerce, The European AgentLink Perspective*. F. Dignum and C. Sierra, Eds., **LNCS (1991)**: 126-147, Springer Verlag, 2001.

[Esteva et al., '04] Esteva, M., Rosell, B., Rodriguez-Aguilar, J. A. and Acros, J. L. Ameli: an agent-based middleware for electronic institutions. In Proceedings of the *Third International Joint Conference on Autonomous Agents and Multiagent Systems*, New York, USA, 236-243, IEEE Computer Society, 2004.

[Faratin, '00] Faratin, P. *Automated service negotiation between autonomous computational agents*. PhD Thesis, Dept. of Electronic Engineering, Queen Mary and Westfield College, University of London, 2000.

[Faratin and Klein, '01] Faratin, P. and Klein, M. Automated contract negotiation and execution as a system of constraints. In Proceedings of the *Workshop on Distributed Constraint Reasoning, at IJCAI'01*, Seattle, USA, 2001.

[Fatima et al., '03] Fatima, S., Wooldridge, M. and Jennings, N. R. Optimal agendas for multi-issue negotiation. In Proceedings of the *Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*, Melbourne, Australia, 2003.

[Finin et al., '94] Finin, T., Fritzson, R., McKay, D. and McEntire, R. KQML as an agent communication language. In Proceedings of the *3rd International Conference on Information and Knowledge Management (CIKM'94)*, Gaithersburg, USA, N. Adam, B. Bhargava et al, Eds., 456-463, ACM Press, 1994.

[FIPA, '01] FIPA. *Agent communication language specification*. The Foundation for Intelligent Physical Agents (www.fipa.org), 2001.

[Franklin and Graesser, '97] Franklin, S. and Graesser, A. Is it an agent, or just a program?: A taxonomy for autonomous agents. In Proceedings of the *Workshop on Intelligent Agents III, Agent Theories, Architectures and Languages, at ECAI'96*, Budapest, Hungary, J. Muller, M. Wooldridge et al, Eds., **LNCS (1193)**, 21-35, Springer Verlag, 1997.

[Freire and Botelho, '02] Freire, J. and Botelho, L. Executing explicitly represented protocols. In Proceedings of the *Workshop on Challenges in Open Systems at AAMAS'02*, Bologna, Italy, 2002.

[Freuder, '90] Freuder, E. C. Partial constraint satisfaction. In Proceedings of the *8th AAAI Conference*, Boston, USA, 278-283, 1990.

[Freuder and Wallace, '92] Freuder, E. C. and Wallace, R. J. Partial constraint satisfaction. *Artificial Intelligence*. **58**(1-3): 21-70, 1992.

[Fruhwirth, '98] Fruhwirth, T. Theory and practice of constraint handling rules. *The Journal of Logic Programming*. **37**: 95-137, 1998.

[Genesereth and Fikes, '92] Genesereth, M. R. and Fikes, R. E. *Knowledge interchange format, version 3.0 reference manual*. Technical Report Logic-92-1, Standford University, 1992.

[Grando and Walton, '06] Grando, A. and Walton, C. Mapa: a language for modelling conversations in agent environments. In Proceedings of the *Intelligent Information Processing and Web Mining Conference 2006 (IIPWM'06)*, Ustron, Poland, 2006.

[Heifetz and Ponsati, '04] Heifetz, A. and Ponsati, C. *All in good time*. Working Paper, Department of Economics and Management, The Open University of Israel, 2004.

[Heiskanen et al., '01] Heiskanen, P., Ehtamo, H. and R.P.Hamalainen. Constraint proposal method for computing pareto solutions in multi-party negotiations. *European Joint Operation Research*. **133**(1): 44-61, 2001.

[Henz and Muller, '00] Henz, M. and Muller, T. An overview of finite domain constraint programming. In Proceedings of the *Fifth Conference of the Association of Asia-Pacific Operational Research Societies*, Singapore, 2000.

[Hirayama and Yokoo, '97] Hirayama, K. and Yokoo, M. Distributed partial constraint satisfaction problem. In Proceedings of the *Third International Conference on Principles and Practice of Constraint Programming (CP'97)*, Linz, Austria, G. Smolka, Ed., **LNCS (1330)**, 222-236, Springer Verlag, 1997.

[Jouvin and Hassas, '02] Jouvin, D. and Hassas, S. Role delegation as multi-agent oriented dynamic composition. In Proceedings of the *Net Object Days (NOD), AgeS Workshop*, Erfurt, Germany, 2002.

[Jung and Tambe, '05] Jung, H. and Tambe, M. On communication in solving distributed constraint satisfaction problems. In Proceedings of the *4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'05)*, Budapest, Hungary, M. Pechouchek, P. Petta et al, Eds., **LNAI (3690)**, 418-429, Springer Verlag, 2005.

[Lambert and Robertson, '05] Lambert, D. and Robertson, D. Matchmaking multi-party interactions using historical performance data. In Proceedings of the *Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Utrecht, The Netherlands, 611-617, 2005.

[Lesser, '99] Lesser, V. R. Cooperative Multiagent systems: A personal view of the state of the art. *IEEE Transactions on Knowledge and Data Engineering*. **11**(1), 1999.

[Lind, '01] Lind, J. Specifying agent interaction protocols with standard UML. In Proceedings of the *2nd International Workshop on Agent-Oriented Software Engineering (AOSE-2001)*, Montreal, Canada, M. Wooldridge, G. Weib et al, Eds., **LNCS (2222)**, 136-147, Springer Verlag, 2001.

[Luo et al., '03] Luo, X., Jennings, N. R., Shadbolt, N., Leung, H. F. and Lee, J. A fuzzy constraint based model for bilateral, multi-issue negotiations in semi-competitive environments. *Artificial Intelligence*. **148**: 53-102, 2003.

[Luo et al., '00] Luo, X., Leung, H. and Lee, J. H. A multi-agent framework for meeting scheduling using fuzzy constraints. In Proceedings of the *Fourth International Conference on Multi-Agent Systems (ICMAS-2000)*, 2000.

[Macho-Gonzales et al., '00] Macho-Gonzales, S., Torrens, M. and Faltings, B. A multi-agent recommender system for planning meetings. In Proceedings of the *Workshop on Agent-based Recommender Systems (WARS' 2000)*, Barcelona, Spain, 2000.

[Malone and Crawston, '94] Malone, T. W. and Crawston, K. The interdisciplinary study of coordination. *ACM Computing Surveys (CSUR)*. **26**(1): 87-119, 1994.

[McGinnis and Robertson, '04] McGinnis, J. and Robertson, D. Realising agent dialogues with distributed protocols. In Proceedings of the *International Workshop on Agent Communication (AC'04)*, New York, USA, R. M. v. Eijk, M.-P. Huget et al, Eds., **LNCS (3396)**, 106-119, Springer, 2004.

[McGinnis and Robertson, '05] McGinnis, J. and Robertson, D. Dynamic and distributed interaction protocols. *Adaptive Agents and Multi-Agent Systems II: Adaptation and Multi-Agent Learning*. D. Kudenko, D. Kazakov et al, Eds., **LNCS (3394)**: 167-184, Springer, 2005.

[McGinnis et al., '03] McGinnis, J., Robertson, D. and Walton, C. Using distributed protocols as an implementation of dialogue games. In Proceedings of the *First European Workshop on Multi-Agent Systems (EUMAS'03)*, Oxford, UK, 2003.

[McGinnis, '06] McGinnis, J. P. *On the mutability of protocols*. PhD Thesis, CISA, School of Informatics, University of Edinburgh, 2006.

[Meisels, '04] Meisels, A. Distributed constraints satisfaction algorithms, performance, communication. In Proceedings of the *10th International Conference on Principles and Practice of Constraint Programming (CP'04)*, Toronto, Canada, 2004.

[Meisels and Kaplansky, '02] Meisels, A. and Kaplansky, E. Distributed timetabling problems (DisTTP). In Proceedings of the *4th International Conference on Practice and Theory of Automated Timetabling*, Gent, Belgium, E. K. Burke and P. D. Causmaecker, Eds., **LNCS (2740)**, Springer, 2002.

[Meisels et al., '02] Meisels, A., Kaplansky, E., Razgon, I. and Zivan, R. Comparing performance of distributed constraints processing algorithms. In Proceedings of the *Workshop on Distributed Constraint Reasoning, at AAMAS'02*, Bologna, Italy, 2002.

[Meseguer et al., '03] Meseguer, P., Bouhmala, N., Bouzoubaa, T., Irgens, M. and Sanchez, M. Current approaches for solving over-constrained problems. *Constraints*. **8**: 9-39, 2003.

[Modi and Veloso, '04] Modi, P. J. and Veloso, M. Multiagent meeting scheduling with rescheduling. In Proceedings of the *Fifth Workshop on Distributed Constraint Reasoning (DCR)*, 2004.

[Modi and Veloso, '05] Modi, P. J. and Veloso, M. Bumping strategies for the multiagent agreement problem. In Proceedings of the *Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, Utrecht, The Netherlands, 2005.

[Noriega, '97] Noriega, P. *Agent-mediated auctions: The fishmarket metaphor*. PhD Thesis, Institut d'Investigacio en Intelligencia Artificial (IIIA), Spain, 1997.

[Norlander, '04] Norlander, T. E. *Constraint relaxation techniques and knowledge base reuse*. PhD Thesis, University of Aberdeen, UK, 2004.

[Norlander et al., '03] Norlander, T. E., Brown, K. N. and Sleman, D. Constraint relaxation techniques to aid the reuse of knowledge bases and problem solvers. In Proceedings of the *23rd SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, Cambridge, UK, pg. 323-335, 2003.

[Nwana et al., '96] Nwana, H. S., Lee, L. and Jennings, N. R. Co-ordination in software agent systems. *BT Technology Journal*. **14**(4): 79-88, 1996.

[Odell et al., '00] Odell, J., Parunak, H. and Bauer, B. Extending uml for agents. In Proceedings of the *Agent-Oriented Information Systems Workshop, at the 17th National Conference on Artificial Intelligence*, Austin, TX, USA, 2000.

[Odell et al., '03] Odell, J., Parunak, H. V. D. and Fleischer, M. Modeling agents and their environment: the communication environment. *Journal of Object Technology*. **2**(3): 39-52, 2003.

[Omicini and Ossowski, '03] Omicini, A. and Ossowski, S. Objective versus subjective coordination in the engineering of agent systems. *Intelligent Information Agents*. M. Klusch, Ed., **LNAI (2586)**: 179-202, Springer Verlag, 2003.

[Osman et al., '06] Osman, N., Robertson, D. and Walton, C. D. Run-time model checking of interaction and deontic models for multi-agent systems. In Proceedings of the *Fifth International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS'06)*, Hakodate, Japan, 2006.

[Paula et al., '00] Paula, G. E. d., Ramos, F. S. and Ramalho, G. L. Bilateral negotiation model for agent-mediated electronic commerce. In Proceedings of the *Agent-*

Mediated Electronic Commerce Workshop (AMEC'00), Barcelona, Spain, F. Dignum and U. Cortes, Eds., **LNAI (2003)**, 1-14, Springer Verlag, 2000.

[Paurobally et al., '03] Paurobally, S., Cunningham, J. and Jennings, N. R. Ensuring consistency in the joint beliefs of interacting agents. In Proceedings of the *Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*, Melbourne, Australia, 2003.

[Pruitt, '81] Pruitt, D. *Negotiation behaviour*. Academic Press, New York, 1981.

[Ramchurn et al., '04] Ramchurn, S. D., Huynh, D. and Jennings, N. R. Trust in multi-agent systems. *The Knowledge Engineering Review*. **19**(1): 1-25, 2004.

[Rich and Knight, '91] Rich, E. and Knight, K. *Artificial Intelligence (2nd edition)*. McGraw-Hill, 1991.

[Robertson, '03] Robertson, D. *Distributed agent protocols*. Technical Report (contact author for details: dr@inf.ed.ac.uk), University of Edinburgh, 2003.

[Robertson, '04a] Robertson, D. A lightweight coordination calculus for agent social norms. In Proceedings of the *Declarative Agent Languages and Technologies at AAMAS'04*, New York, USA, J. Leite, A. Omini et al, Eds., **LNCS (3476)**, Springer Verlag, 2004a.

[Robertson, '04b] Robertson, D. A lightweight method for coordination of agent oriented web services. In Proceedings of the *AAAI Spring Symposium on Semantic Web Services*, California, USA, 2004b.

[Robertson, '04c] Robertson, D. Multi-agent coordination as distributed logic programming. In Proceedings of the *20th International Conference on Logic Programming*, Saint-Malo, France, B. Demoen and V. Lifschitz, Eds., **LNCS (3132)**, Springer Verlag, 2004c.

[Rosenschein and Zlotkin, '94] Rosenschein, J. S. and Zlotkin, G. *Rules of encounter: Desigining conventions for automated negotiation among computers*. MIT Press, Cambridge, MA, 1994.

[Rudova and Matyska, '99] Rudova, H. and Matyska, L. Uniform framework for solving over-constrained and optimisation problems. In Proceedings of the *Post-Conference Workshop on Modelling and Solving Soft Constraints*, Alexandria, Virginia, USA, 1999.

[Ruskey and Weston, '05] Ruskey, F. and Weston, M. A survey of Venn diagrams: generalizations and extensions. *The Electronic Journal of Combinatorics, www.combinatorics.org/Surveys/ds5/VennOtherEJC.html*, 2005.

[Schiex, '92] Schiex, T. Possibilistic constraint satisfaction problem or "How to handle soft constraints?" In Proceedings of the *8th International Conference on Uncertainty in Artificial Intelligence*, Stanford, California, 268-275, 1992.

[Schulte and Carlson, '06] Schulte, C. and Carlson, M. Finite domain constraint Programming systems. *Handbook of constraint programming*. F. Rossi, P. V. Beek et al, Eds., Elsevier, 2006.

[Schumacher and Ossowski, '06] Schumacher, M. and Ossowski, S. The governing environment. In Proceedings of the *2nd International Workshop on Environments for Multiagent Systems (E4MAS'05), at AAMAS'05*, Utrecht, The Netherlands, D. Weyns, H. V. D. Parunak et al, Eds., **LNAI (3830)**, 88-104, Springer Verlag, 2006.

[Searle, '69] Searle, J. *Speech acts*. Cambridge University Press, Cambridge, 1969.

[SICS, '99] SICS. *SICStus Prolog User's Manual*. Swedish Institute of Computer Science (SICS) (http://www.sics.se/sicstus.html), Stockholm, 1999.

[Sycara, '98] Sycara, K. P. Multiagent systems. *AI Magazine*. **19**(2): 79-92, 1998.

[Tsang, '93] Tsang, E. *Foundations of constraint satisfaction*. Academic Press, 1993.

[Tsuruta and Shintani, '00] Tsuruta, T. and Shintani, T. Scheduling meetings using distributed values constraint satisfaction. In Proceedings of the *European Conference on Artificial Intelligence (ECAI)*, Germany, 2000.

[Walton, '04a] Walton, C. D. Model checking multi-agent web services. In Proceedings of the *AAAI Spring Symposium on Semantic Web Services*, Stanford, California, USA, 2004a.

[Walton, '04b] Walton, C. D. Multi-agent dialogue protocols. In Proceedings of the *8th International Symposium on Artificial Intelligence and Mathematics*, Ft. Lauderdale, Florida, 2004b.

[Walton and Barker, '04] Walton, C. D. and Barker, A. An agent-based e-science experiment builder. In Proceedings of the *First International Workshop on Semantic Intelligent Middleware for the Web and the Grid, at ECAI'04*, Valencia, Spain, 2004.

[Walton and Robertson, '02] Walton, C. D. and Robertson, D. *Flexible multi-agent protocols*. Technical Report EDI-INF-RR-0164, University of Edinburgh, 2002.

[Weib, '01] Weib, G. Congnition, sociability, and constraints. *Balancing Reactivity and Social Deliberation in Multi-Agent Systems: From RoboCup to Real-World*

Applications. M. Hannebauer, J. Wendler et al, Eds., **LNCS (2103)**, Springer Verlag, 2001.

[Wooldridge, '00] Wooldridge, M. Semantic issues in the verification of agent communication languages. *Autonomous Agents and Multi-Agent Systems*. **3**(1): 9-31, 2000.

[Wooldridge, '02] Wooldridge, M. *An introduction to multiagent systems*. MIT Press, 2002.

[Wooldridge and Jennings, '95] Wooldridge, M. and Jennings, N. R. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*. **10**: 115-152, 1995.

[Yang and Fong, '92] Yang, Q. and Fong, P. *Solving partial constraint satisfaction problems using local search and abstraction*. Technical Report CS-92-50, University of Waterloo, Canada, 1992.

[Yokoo, '93] Yokoo, M. Dynamic variable/value ordering heuristics for solving large-scale distributed constraint satisfaction problems. In Proceedings of the *12th International Workshop on Distributed Artificial Intelligence*, Hidden Valley, Pennsylvania, USA, 407-422, 1993.

[Yokoo, '01] Yokoo, M. *Distributed constraint satisfaction: foundations of cooperation in multi-agent systems*. Springer Verlag, 2001.

[Yokoo et al., '98] Yokoo, M., Durfee, E. H., Ishida, T. and Kubawara, K. The distributed constraint satisfaction problem: formalization and algorithm. *IEEE Transactions on Knowledge and Data Engineering*. **10**(5): 673-685, 1998.

[Yokoo and Hirayama, '93] Yokoo, M. and Hirayama, K. Distributed partial constraint satisfaction problem. In Proceedings of the *Third International Conference on Principles and Practice of Constraint Programming*, **LNCS (1330)**, 222-236, Springer Verlag, 1993.

[Zhou et al., '05] Zhou, L., Sattar, A. and Goodwin, S. Handling over-constrained problems in distributed multi-agent systems. In Proceedings of the *18th Conference of the Canadian Society for Computational Studies of Intelligence (Canadian AI 2005)*, Victoria, Canada, B. Kegl and G. Lapalme, Eds., **LNAI (3501)**, 13-24, Springer Verlag, 2005.